Thèse présentée par

Fanny Pascual

pour obtenir le titre de

DOCTEUR de L'UNIVERSITÉ D'ÉVRY VAL D'ESSONNE

Spécialité : **Informatique**

Optimisation dans les réseaux : de l'approximation polynomiale à la théorie des jeux

Date de soutenance : 12 octobre 2006

Composition du jury

Rapporteurs	Michael Bender
	Michel de Rougemont
Examinateurs	Michel Habib
	Elias Koutsoupias
Directeur de thèse	Evripidis Bampis
Co-encadrant	Eric Angel

Thèse préparée au Laboratoire Informatique, Biologie Intégrative et Systèmes Complexes
de l'université d'Évry Val d'Essonne - FRE 2873 du CNRS

Table des matières

Introduction

Les réseaux occupent une place prépondérante dans notre vie : que ce soient les réseaux informatiques, comme internet, les réseaux de télécommunication, comme les réseaux optiques, ou bien encore le réseau routier. Bien qu'a priori très différents, ces réseaux ont une structure similaire, la définition générale d'un réseau étant en effet "un ensemble de nœuds liés par des canaux". La plupart des réseaux ont alors en commun le fait de pouvoir être modélisés de la même façon : à base de graphes et/ou d'ordonnancements (le graphe représente la structure générale du réseau, alors que l'ordonnancement permet souvent de faire un "zoom" sur un canal, ou lien). Les méthodes qui permettent d'obtenir de bonnes performances sur ces réseaux sont donc souvent génériques. Dans cette thèse, nous nous intéressons à des problèmes d'optimisation motivés par différentes applications liées aux réseaux. La *recherche opérationnelle* et *l'algorithmique* (et notamment *l'optimisation combinatoire*) sont des domaines qui nous permettent d'étudier les performances d'un réseau donné, et de concevoir un réseau ou de le modifier afin d'améliorer ses performances pour un critère donné. Ce critère peut être la minimisation du coût de transmission des paquets dans un réseau optique, la minimisation de la distance parcourue par un véhicule dans un réseau routier, ou bien encore la minimisation du temps moyen (ou du temps maximum) de parcours d'un paquet dans un réseau informatique. Notre but dans cette thèse est donc d'étudier de tels problèmes, et de concevoir autant que possible des algorithmes qui induisent de bonnes performances globales.

Il est important de pouvoir résoudre chaque problème en un temps raisonnable. Le nombre de solutions possibles pour chacun de ces problèmes étant fini (nous sommes face à des problèmes d'optimisation combinatoire), nous pouvons bien sûr examiner une à une les solutions et ensuite sélectionner la meilleure. Ceci n'est pas réalisable en pratique car le nombre de solutions possibles "explose" avec la taille de l'instance du problème : le temps pour résoudre le problème est alors exponentiel en la taille de l'instance. Il est communément admis qu'un algorithme s'éxécute en un temps raisonnable seulement si son temps d'éxécution est polynomial en la taille de l'instance. Il est cependant fortement conjecturé que certains problèmes ne sont résolvables qu'en temps exponentiel en la taille de l'instance : c'est le sens de la célèbre conjecture $P \neq NP$ (P représente les problèmes résolvables en un temps polynomial, et NP ceux résolvables en un temps exponentiel). Si $P \neq NP$ alors il existe des problèmes que l'on ne peut résoudre qu'en un temps exponentiel : ces problèmes sont dits NP-difficiles. Les problèmes auxquels nous faisons face dans cette thèse sont NP-difficiles. Nous cherchons alors à trouver, en un temps polynomial, des solutions approchées aux problèmes.

Dans certains réseaux informatiques (comme par exemple internet), il n'est pas possible d'avoir un algorithme centralisé pour résoudre nos problèmes d'optimisation. En effet, la structure et la taille importante de ces réseaux font qu'il n'est pas envisageable d'avoir un algorithme centralisé.

Nous cherchons alors à avoir des algorithmes totalement décentralisés (par exemple dans lesquels chaque lien qui route des paquets a une *politique locale*, i.e. un algorithme qui lui est propre et qui lui permet de donner un ordre aux paquets qu'il route). D'autre part, ces réseaux sont utilisés par des utilisateurs indépendants et individualistes : chacun de ces utilisateurs possède son propre critère (fonction objectif) qu'il souhaite optimiser. Par exemple un utilisateur possédant un paquet à router dans un réseau souhaite minimiser la date d'arrivée de son paquet. Ces utilisateurs ne doivent pas nécessairement obéir à un protocole mais possèdent une certaine *liberté d'action* qui n'existe pas dans les problèmes classiques d'optimisation combinatoire. Cette liberté d'action peut prendre plusieurs formes. Les utilisateurs peuvent par exemple choisir d'effectuer certaines actions (par exemple choisir le chemin à emprunter dans le cas où l'utilisateur désire router son paquet). Il se peut également que les utilisateurs doivent faire face à un protocole gérant le réseau, mais sont les seuls à connaître leurs propres caractéristiques. Le protocole (ou algorithme) devant retourner une solution, les utilisateurs doivent lui indiquer leurs caractéristiques, en fonction desquelles le protocole construit une solution. Ils peuvent alors manipuler le protocole en lui procurant de fausses informations si cela améliore leur fonction objectif personnelle. Par exemple, un utilisateur possédant un paquet (de données), qu'il souhaite router dans un réseau, peut mentir sur la taille de son paquet – en y ajoutant par exemple des données inutiles pour augmenter artificiellement sa taille – si cela permet à son paquet d'être routé plus rapidement. En présence de tels utilisateurs, il n'est pas possible de supposer, comme dans les problèmes classiques d'optimisation combinatoire où les données sont statiques et connues par avance, qu'un algorithme peut imposer une solution à ses utilisateurs.

La *théorie des jeux*, fondée par J. Von Neumann[1], permet de modéliser le comportement d'utilisateurs rationnels. Dans les cas où nous considérons des utilisateurs indépendants et individualistes, nos travaux s'inscrivent plus particulièrement dans le domaine de la *théorie des jeux algorithmique*, domaine à l'intersection de l'optimisation combinatoire et de la théorie des jeux, et qui s'est considérablement développé ces cinq dernières années. Puisque les utilisateurs sont indépendants et se comportent chacun de manière à optimiser leur fonction objectif, nous supposons que la situation que l'on obtient avec de tels utilisateurs est telle qu'aucun utilisateur ne peut améliorer sa fonction objectif en se comportant autrement (étant donnés le protocole, les comportements des autres utilisateurs, et les actions qu'il lui est possible d'entreprendre). Une telle situation est alors un équilibre de Nash (rendu célèbre par J. Nash en 1951 [63]). Cette situation peut être très éloignée d'une solution optimale vis-à-vis d'un critère de qualité global du réseau. Ce critère (ou fonction objectif globale) peut par exemple être la minimisation de la date d'arrivée maximale (ou moyenne) d'un paquet à sa destination. Notre but est alors d'évaluer la performance d'un protocole donné face à des agents individualistes, et de concevoir des protocoles avec lesquels la fonction objectif globale est la meilleure possible, malgré le comportement individualiste des utilisateurs. Dans le cas où les agents doivent indiquer leurs caractéristiques au protocole, nous cherchons des algorithmes avec *véracité garantie*, i.e. tels que les agents ne sont pas tentés de mentir, car ils maximisent leur propre fonction objectif en déclarant la vérité au protocole. Le protocole connaît alors les données réelles du problème, et trouve ainsi une solution en connaissance de cause.

Les résultats exposés dans cette thèse sont de deux natures : ce sont tout d'abord des résul-

[1] Von Neumann a montré le premier résultat important de la théorie des jeux en 1928, et a écrit avec O. Morgenstern le livre fondateur de cette théorie en 1944 [62].

tats d'impossibilité, dus à des propriétés de complexité (si $P \neq NP$ nous ne pouvons pas obtenir d'algorithme exact en un temps polynomial), ou à des propriétés structurelles (par exemple si nous souhaitons uniquement un algorithme complètement distribué, ou bien si nous souhaitons uniquement un algorithme avec véracité garantie). Le deuxième type de résultats concerne des résultats constructifs : nous donnons des algorithmes (souvent approchés) pour résoudre les problèmes étudiés.

Qu'il y ait ou non des utilisateurs indépendants et individualistes dans le réseau, nous nous intéressons particulièrement aux *algorithmes avec garanties de performance*. Ces algorithmes, introduits par R. Grham [41] en 1966, garantissent une certaine qualité des solutions retournées : les solutions retournées par un algorithme ρ-approché sont en effet toutes au pire ρ fois moins bonnes que les solutions optimales pour le critère considéré (ρ est alors un rapport d'approximation de l'algorithme). Les algorithmes avec garantie de performance présentent plusieurs avantages : ils permettent bien sûr d'être sûr d'avoir une garantie sur la qualité d'une solution (et ce qu'elle que soit l'instance), ce qui est appréciable en pratique. Ceci engendre également, d'un point de vue théorique, un défi puisque l'enjeu est de proposer des algorithmes qui s'approchent le plus possible de l'optimal (on explore alors les limites de ce qu'il est possible de faire pour résoudre ces problèmes en un temps polynomial). Enfin, un algorithme $\mathcal{A}$ avec garantie de performance a une autre qualité sur le plan pratique : il permet de garantir la qualité d'une solution obtenue avec un autre algorithme $\mathcal{B}$ (qui lui n'est pas nécessairement avec garantie de performance), en la comparant avec la solution retournée par $\mathcal{A}$ sur la même instance (cette solution permet en effet de donner une borne – inférieure pour les problèmes de minimisation, supérieure pour les problèmes de maximisation – sur la valeur de la fonction objectif dans une solution optimale).

Contribution

Les chapitres 2, 3 et 7 se rapportent à des réseaux modélisés par des graphes ; les chapitres 4, 5 et 6 se rapportent à des réseaux modélisés par des ordonnancements. Les différentes problématiques étudiées sont plus ou moins ciblées sur des applications pratiques. Ainsi les chapitres 2 et 3 (et notamment le chapitre 2) traitent de problématiques très spécifiques, les autres étant peut-être plus génériques. Le **premier chapitre** rappelle rapidement les notions nécessaires pour aborder la suite de ce document. Des pointeurs vers des ouvrages de référence sont indiqués pour chaque sujet (pour certaines notions introduites ces dernières années, nous indiquons les articles définissant ces notions).

Dans le **chapitre 2**, nous étudions un problème de groupage de trafic dans un réseau optique en étoile : un ensemble de nœuds sont reliés à un nœud central par une fibre optique contenant plusieurs longueurs d'onde, et souhaitent émettre des données vers les autres nœuds. Les données peuvent être soit routées optiquement (elles doivent dans ce cas bénéficier d'une longueur d'onde pour elles seules de leur source à leur destination), soit être routées électroniquement (plusieurs données sont alors groupées dans une même longueur d'onde, ce qui nécessite de transformer, au nœud central, le signal optique en signal électronique pour "dégrouper" ou "regrouper" les données). Notre but est ici de maximiser la quantité de données routées optiquement. Ces données n'induiront en effet pas de coût de calcul (et de délai important) ainsi que de coût de stockage en ne nécessitant pas d'être traitées sous forme électronique au nœud central. Nous cherchons alors à déterminer les cas où ce problème est polynomial ou NP-difficile, et à donner pour chacun de ces cas des algorithmes exacts ou approchés avec garantie de performance.

Dans le **chapitre 3**, nous étudions le problème NP-difficile de tournées de véhicule. Nous avons : un graphe complet dont les nœuds représentent les clients et les poids sur les arêtes entre chaque couple de clients représentent la distance physique séparant ces clients ; une quantité de biens demandés par chaque client ; et un véhicule ayant une capacité C (i.e. qui peut servir une quantité de biens égale à C avant de retourner au dépôt). Le but est de minimiser la distance totale que le véhicule doit couvrir afin de servir tous les clients et de retourner au dépôt. La recherche locale donne souvent de bons résultats pour ce genre de problème. Nous nous intéressons alors à un voisinage de taille exponentielle dont nous cherchons à montrer qu'il est explorable en un temps polynomial.

Dans le **chapitre 4**, nous étudions le problème qui consiste à ordonnancer des tâches de longueurs différentes sur des machines (ou liens) parallèles identiques. Nous nous intéressons à des mesures d'équité dans ce problème d'ordonnancement : équité entre les machines (on veut minimiser le nombre moyen de tâches non terminées par machine et par unité de temps) et équité entre les tâches (on suppose que le but de chaque tâche est de se terminer le plus rapidement possible). Nous nous intéressons plus particulièrement à ces mesures d'équité parmi l'ensemble des ordonnancements qui minimisent la date de fin moyenne des tâches.

Jusqu'à présent, nous avons considéré des algorithmes centralisés, et dans lesquels les données sont connues. Dans le dernier problème mentionné, les tâches souhaitaient minimiser leurs dates de fin, mais n'avaient aucun moyen de le faire, puisque la solution était imposée par un algorithme centralisé. Nous supposons maintenant que certaines données du problème (par exemple les tâches dans un problème d'ordonnancement) sont détenues par des agents indépendants ayant leurs propres objectifs (par exemple minimiser la date à laquelle leur tâche est ordonnancée) et ayant une certaine liberté d'action. Ils se comporteront alors de manière à atteindre leur objectif, quelles qu'en soient les conséquences pour les autres utilisateurs. Comme nous l'avons dit précédemment, ces agents seront considérés comme rationnels, et leur comportement sera modélisé grâce à la théorie des jeux.

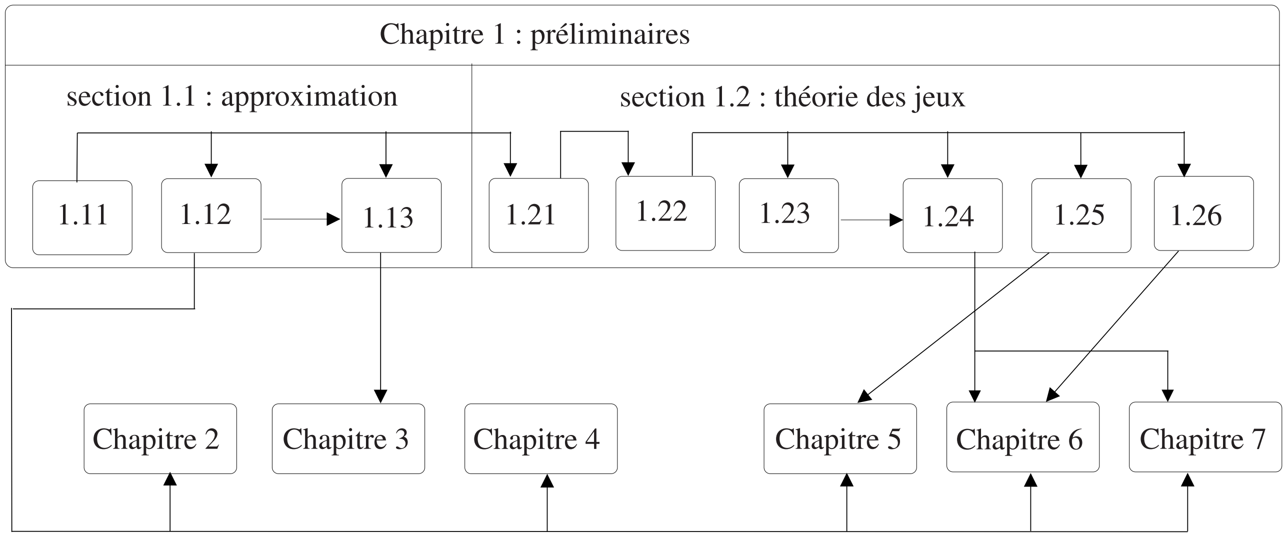
Dans le **chapitre 5**, nous nous intéressons à un problème d'ordonnancement dans lequel des tâches indépendantes souhaitent être ordonnancées sur des machines parallèles, et chaque tâche souhaite minimiser sa date de fin. Notre but est de concevoir un protocole (un algorithme) qui propose un ordonnancement de ces tâches de manière à minimiser la date de fin de l'ordonnancement (la date à laquelle toutes les tâches ont été exécutées). Chaque machine a une politique (algorithme local) qui lui indique l'ordre dans lequel exécuter ses tâches. Les tâches sont autonomes : chaque tâche peut choisir elle-même la machine sur laquelle elle sera ordonnancée, et peut donc refuser d'aller sur la machine proposée par le protocole. Le protocole doit donc retourner un ordonnancement qui non seulement a une date de fin la plus petite possible, mais qui est également stable, i.e. tel que chaque tâche n'ait pas (ou peu) intérêt à aller sur une machine sur laquelle elle n'a pas été affectée. Nous étudions alors le compromis entre la stabilité de l'ordonnancement et son rapport d'approximation. Nous étudions en particulier le cas où chaque machine ordonnance ses tâches par ordre de longueurs décroissantes, celui où elle ordonnance ses tâches par longueurs croissantes, ainsi qu'une politique probabiliste.

Dans le **chapitre 6**, nous considérons des algorithmes avec véracité garantie pour ordonnancer des tâches indépendantes. Ici, nous supposons que la longueur d'une tâche (i.e. sa durée d'exécution) est connue uniquement par la tâche. Le but de chaque tâche est toujours de minimiser sa date de fin, et une tâche peut donc mentir sur sa longueur à l'algorithme, si cela peut diminuer sa date de fin. Une tâche peut en effet artificiellement augmenter sa longueur sans que l'algorithme puisse le détecter. Notre but est de donner des bornes inférieures sur le meilleur rapport d'approximation qu'un algorithme avec véracité garantie peut avoir. Nous souhaitons également donner des algorithmes avec véracité garantie dont le rapport d'approximation se rapproche le plus possible de la borne inférieure. Nous nous intéressons également dans ce chapitre aux mécanismes de coordination avec véracité garantie : dans ce cas, une tâche a une plus grande liberté d'action encore puisqu'elle choisit elle-même la machine sur laquelle elle s'exécute, sa longueur étant par ailleurs toujours connue d'elle seule.

Dans le **chapitre 7**, nous nous intéressons au routage de paquets dans un réseau "store and forward" dont la topologie est un chemin, un arbre, ou bien un anneau. Nous considérons là aussi des mécanismes de coordination, c'est-à-dire qu'il n'y a pas d'algorithme centralisé (qui indiquerait à chaque paquet le chemin à prendre, et à chaque lien quel paquet router et quand le router), mais chaque lien du réseau a une politique locale qui lui indique comment router ses paquets. Les paquets partent d'une même source, et chaque paquet a une taille et une destination. Ils sont autonomes et choisissent eux-mêmes leur route, sachant que leur but est d'arriver le plus tôt possible à leurs destinations. Nous étudions l'impact de plusieurs politiques (basées sur la taille des paquets et/ou la distance à leur destination) sur les deux fonctions objectifs suivantes : la date à laquelle tous les paquets sont arrivés, et la date moyenne d'arrivée des paquets à leur destination.

Plan de lecture de ce mémoire

Lors de la lecture des chapitres 2 à 7, il est utile de connaître les notions introduites dans le premier chapitre. Les chapitres 2 à 7 peuvent ensuite être lus indépendamment les uns des autres. Nous avons indiqués à la fin de chaque chapitre une conclusion et des perspectives relatives au chapitre. Une conclusion générale se trouve également à la fin du document. Le schéma suivant illustre les dépendances entre les différentes parties de ce mémoire.



Chapitre 1

Préliminaires

Nous présentons dans ce chapitre les principales notions utiles pour exposer les travaux réalisés dans cette thèse. Il s'agit de définitions standards, s'inscrivant dans le cadre de l'optimisation combinatoire (et plus particulièrement de l'approximation) et de la théorie des jeux (de la théorie des jeux algorithmique en particulier). Nous fixons également dans ce chapitre la terminologie et les notations utilisées dans la suite de ce document.

1.1 Optimisation combinatoire et approximation

Un problème d'optimisation combinatoire consiste à trouver la meilleure solution (suivant un critère donné) parmi un ensemble fini de solutions. Ainsi, un *problème d'optimisation combinatoire* Π comprend un ensemble d'instances D_Π. Chaque instance $I \in D_\Pi$ est une paire $(S_\Pi(I), f_\Pi(., I))$, où $S_\Pi(I)$ est un ensemble fini de solutions réalisables et $f_\Pi(., I)$ est une fonction de coût qui associe à chaque solution $s \in S_\Pi(I)$ son coût $f_\Pi(s, I) \in \mathbb{R}$. Le problème consiste à trouver, pour un problème de minimisation (respectivement maximisation), une solution $s^* \in S_\Pi(I)$ telle que $f_\Pi(s^*, I) \leq f_\Pi(x, I)$ (respectivement $f_\Pi(s^*, I) \geq f_\Pi(x, I)$) pour tout $x \in S_\Pi(I)$. Une telle solution s^* est appelée un *optimum global* de l'instance.

Un *problème de décision* Π est défini par un ensemble d'instances D_Π et par une question à laquelle la réponse est soit "oui" soit "non". L'ensemble des instances est partitionné en un ensemble D^+ d'instances positives (celles pour lesquelles la réponse à la question est oui) et un ensemble D^- d'instances négatives (celles pour lesquelles la réponse à la question est non). Étant donnée une instance $I \in D_\Pi$, il s'agit de déterminer si la réponse à la question est oui ou non. À chaque problème d'optimisation combinatoire peut être associé un problème de décision : étant donné un nombre k et une instance I, existe-t-il une solution $s \in S_\Pi(I)$ telle que $f_\Pi(s, I) \leq k$ pour un problème de minimisation (ou $f_\Pi(s, I) \geq k$ pour un problème de maximisation) ?

Nous nous intéressons à la *complexité en temps*, i.e. le nombre d'étapes nécessaires pour résoudre un problème. La classe P est l'ensemble des problèmes de décision pouvant être résolus en temps polynomial par une machine de Turing déterministe, et la classe NP est l'ensemble des problèmes de décision pouvant être résolus en temps polynomial par une machine de Turing non déterministe. Un programme informatique pouvant être modélisé par une machine de Turing déterministe, il sera possible d'avoir un algorithme pour résoudre le problème étudié en un temps polynomial si ce problème (ou le problème de décision associé) appartient à la classe P. Il est aujourd'hui fortement conjecturé que $P \neq NP$. Si cette conjecture s'avère vraie, alors il n'est pas possible de trouver en un temps polynomial l'optimum global d'une instance pour un problème NP-difficile. Il est alors utile de s'intéresser à des algorithmes qui retournent en un temps raisonnable (polynomial) des solutions dont le coût est le plus proche possible du coût de l'optimum global. Les sections 1.1.2 et 1.1.3 présentent deux domaines liés à cette question.

Nous présentons tout d'abord dans la section suivante quelques définitions et notations relatives à des problèmes d'ordonnancement et qui seront utiles par la suite. Nous illustrerons notamment chaque notion introduite dans la suite de ce chapitre grâce à un problème d'ordonnancement. Les notations introduites dans cette section seront également utilisées dans les chapitres 4, 5 et 6 traitants des problèmes d'ordonnancements.

1.1.1 Ordonnancements : rappels et notations

Les problèmes d'ordonnancements sont très nombreux et variés (voir [26, 48, 28] pour une vue d'ensemble sur le sujet). Le problème d'ordonnancement le plus courant est peut être l'exécution de processus sur des processeurs parallèles, mais de nombreux problèmes (en recherche opérationnelle notamment) peuvent être modélisés par des problèmes d'ordonnancements. Dans un réseau, le routage de tâches (ou paquets) sur des liens parallèles peut ainsi être modélisé par un problème d'ordonnancement.

Nous considérons des problèmes d'ordonnancement dans lesquels on a un ensemble de tâches et un ensemble de m machines parallèles. Chaque tâche i doit être ordonnancée (ou exécutée) pendant un temps l_i sur exactement une des m machines, chacune d'elles pouvant exécuter au plus une tâche en même temps. Dans la suite de ce document, chaque tâche i est caractérisée par sa longueur (ou durée d'exécution) $l_i > 0$ et par un numéro d'identification i (un entier différent pour chacune des tâches). Ainsi les tâches sont toujours comparables deux à deux. Les machines sont identiques : la durée d'exécution d'une tâche est la même quelque soit la machine sur laquelle elle est ordonnancée[1]. Chacune des machines (quelquefois aussi appellées processeurs) a un numéro d'identification, allant de 1 à m, et la $i^{\text{ème}}$ machine est notée P_i.
Étant donné un ordonnancement nous noterons C_j la *date de fin* de la tâche j : c'est la plus petite date à laquelle la tâche j a été complètement exécutée (C_j est donc égale à la date de début d'exécution de la tâche j, plus l_j). De même, la date de fin d'une machine est égale à la date de fin de la dernière tâche ordonnancée sur cette machine. La date de fin d'un ordonnancement (ou "makespan" en anglais) est égale à la date de fin maximale d'une machine de cet ordonnancement : c'est donc également la date de fin de la dernière tâche de cet ordonnancement. Nous noterons généralement C_{max} la date de fin d'un ordonnancement. Dans la suite de ce document nous considérerons que la date de début d'un ordonnancement est la date 0. La figure 1.1 montre un exemple d'ordonnancement. Par convention, et sauf mention contraire, le numéro à l'intérieur d'une tâche correspondra, tout au long de ce document, à la longueur de la tâche.

Le problème qui consiste à minimiser la date de fin d'une machine, introduit par R. Graham [41] et noté $(P||C_{max})$ quand les machines sont identiques, est un problème NP-difficile [40].

[1] Nous ferons quelques fois allusion au cas où les machines ne sont plus identiques mais ont des vitesses d'exécution différentes : la durée d'exécution d'une tâche sur une machine est alors égale à la longueur de la tâche divisée par la vitesse de la machine

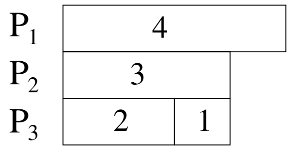


FIG. 1.1 – Exemple d'ordonnancement. La date de fin de la tâche de longueur 1 est 3, et la date de fin de l'ordonnancement est 4.

Étant donné une liste de tâches (un ensemble de tâches triées selon un certain critère) un algorithme de liste pour ce problème est un algorithme qui ordonnance les tâches de la liste de manière gloutonne : il ordonnance les tâches de la liste une à une, en commençant par la première tâche de la liste, et en plaçant dès qu'une machine est inactive (elle n'exécute pas de tâche) une tâche sur cette machine. Nous nous référerons par la suite à *l'algorithme de liste LPT* (pour "Longest Processing Time First"), qui consiste à ordonner les tâches par longueurs décroissantes (et par indices décroissants si plusieurs tâches ont la même longueur) et à ordonnancer ensuite chacune de ces tâches de manière gloutonne : dès qu'une machine devient inactive, elle exécute la plus grande tâche non encore ordonnancée. De même, *l'algorithme de liste SPT* (pour "Shortest Processing Time First"), consiste à ordonner les tâches par longueurs croissantes (ou par indices croissants si plusieurs tâches ont la même longueur) et à ordonnancer ensuite chacune de ces tâches de manière gloutonne. Un ordonnancement SPT (respectivement LPT) est un ordonnancement retourné par l'algorithme de liste SPT (respectivement LPT), aussi plus simplement appelé algorithme SPT (respectivement LPT).

1.1.2 Approximation polynomiale avec garantie de performance

Si l'on souhaite un algorithme qui retourne en un temps polynomial de bonnes solutions à un problème d'optimisation combinatoire NP-difficile, nous avons vu que si $P \neq NP$, cet algorithme ne retournera pas nécessairement toujours la meilleure solution du problème, mais uniquement des solutions approchées. Il est alors utile de connaître la "qualité" des solutions retournées. Le but des algorithmes d'approximation avec garantie de performance est de garantir une certaine qualité des solutions qu'ils retournent. Un algorithme avec garantie de performance est caractérisé par un *rapport d'approximation* ρ qui indique que chaque solution retournée par l'algorithme est au pire ρ fois moins bonne que la meilleure solution.

Ainsi, pour un problème de minimisation, la qualité d'une solution s est mesurée par le rapport ρ entre son coût $f_\Pi(s, I)$ et le coût d'un optimum global $f_\Pi(s^*, I)$: $\rho = f_\Pi(s, I)/f_\Pi(s^*, I)$. Un algorithme est $(1+\varepsilon)$-approché (ou a un rapport d'approximation $1+\varepsilon$) si et seulement si pour chaque instance $I \in D_\Pi$ il fournit une solution s dont le coût est au plus $1+\varepsilon$ fois le coût d'un optimum global, c'est-à-dire $\rho \leq 1+\varepsilon$. Dans ce cas la solution est dite $(1+\varepsilon)$-approchée. Pour un problème de maximisation nous utilisons le rapport d'approximation $\rho = f_\Pi(s^*, I)/f_\Pi(s, I)$. Le rapport d'approximation est donc toujours supérieur ou égal à 1.

Un algorithme ayant un rapport d'approximation égal à ρ est dit ρ-approché. Une famille d'algorithmes $(1+\varepsilon)$-approchés pour tout $\varepsilon > 0$, est appelée un *schéma d'approximation polynomial* (ou PTAS pour "polynomial time approximation scheme") si le temps d'exécution de chaque algorithme est polynomial en fonction de la taille de l'instance.

Par exemple, le rapport d'approximation de l'algorithme de liste SPT (présenté dans la section précédente) pour le problème $(P||C_{max})$ est égal à $2 - \frac{1}{m}$, et le rapport d'approximation de l'algorithme de liste LPT pour ce même problème est égal à $\frac{4}{3} - \frac{1}{3\,m}$.

La définition de rapport d'approximation ci-dessus est valable pour les algorithmes déterministes. Nous pouvons de la même façon avoir un algorithme probabiliste avec garantie de performance. Un algorithme probabiliste pour un problème de minimisation (respectivement maximisation) est $(1 + \varepsilon)$-approché si l'espérance du coût de la solution qu'il retourne est au plus (respectivement au moins) $(1 + \varepsilon)$ fois le coût d'une solution optimale. Nous parlerons alors de l'espérance du rapport d'approximation d'un algorithme probabiliste.

Les livres [48] et [79] sont de bons documents sur les algorithmes d'approximation avec garantie de performance.

1.1.3 Recherche locale

Afin de trouver de bonnes solutions à un problème d'optimisation combinatoire NP-difficile, on utilise également des heuristiques, qui sont des algorithmes qui - bien que ne proposant pas toujours de garantie de performance - retournent en général de bonnes solutions du problème. Une classe importante d'heuristiques est celle des *algorithmes de recherche locale* (voir [1, 16] pour une vue d'ensemble sur ce sujet). La recherche locale est une méthode générale largement utilisée pour résoudre de manière approchée de nombreux problèmes d'optimisation combinatoire NP-difficiles. Nous avons vu qu'un problème d'optimisation combinatoire a un ensemble fini de solutions, et une fonction de coût qui associe à chaque solution une valeur. L'objectif est de trouver un *optimum global*, c'est à dire une solution qui a la plus petite (respectivement grande) valeur pour un problème de minimisation (respectivement maximisation). Un algorithme de recherche locale introduit une structure de voisinage sur l'ensemble des solutions d'un problème d'optimisation : à chaque solution est associé un ensemble de solutions appelées solutions *voisines*. L'algorithme de recherche locale (standard) part alors d'une solution s_0, choisit dans le voisinage de cette solution une solution s_1 strictement meilleure que s_0 (s'il en existe une), examine ensuite le voisinage de s_1 pour obtenir de nouveau une solution strictement meilleure que s_1 (s'il en existe), et ainsi de suite. L'algorithme s'arrête par conséquent lorsqu'il atteint une solution qui ne possède pas de solution strictement meilleure qu'elle dans son voisinage. De telles solutions sont appelées des optima *locaux* : un *optimum local*, est une solution qui n'a pas de solution voisine strictement meilleure qu'elle, c'est à dire ayant un coût plus petit dans le cas d'un problème de minimisation, et ayant un coût plus grand dans le cas d'un problème de maximisation.

En principe, le voisinage d'une solution s'obtient en appliquant des transformations simples sur cette solution. Un voisinage n'a pas besoin d'être symétrique : il est possible qu'une solution s_i ait pour voisine une solution s_j, sans que s_j ait dans son voisinage la solution s_i. En définissant une structure de voisinage $\mathcal{N}_\pi$ sur l'ensemble des solutions d'un problème d'optimisation combinatoire π, on obtient un *problème de recherche locale*, qui est le suivant : étant donné une instance x, trouver une solution localement optimale (par rapport au voisinage $\mathcal{N}_\pi$).

Considérons par exemple le problème d'ordonnancement $(P||C_{max})$: nous avons un ensemble de tâches à ordonnancer sur m machines et nous souhaitons obtenir un ordonnancement dont la date de fin est la plus petite possible. G. Finn et E. Horowitz considèrent dans [38] le voisinage *jump* suivant. Soit un ordonnancement (quelconque) $\mathcal{S}$. Une solution voisine de $\mathcal{S}$ est obtenue en déplaçant une tâche de sa machine vers une autre machine (et en ne changeant pas l'emplacement des autres tâches). Un optimum local est alors une solution dans laquelle aucune tâche ne peut changer (unilatéralement) de machine sans augmenter la date de fin de l'ordonnancement. Les auteurs de [38] montrent que le rapport d'approximation d'une telle solution est de $2 - \frac{2}{m+1}$.

L'algorithme de recherche locale doit choisir à chaque étape, dans le voisinage d'une solution, quelle est la prochaine solution. Il existe deux règles pour effectuer ce choix. Selon la règle de la *plus profonde descente*, l'algorithme de recherche locale choisit la meilleure solution du voisinage (en cas d'égalité le choix se fait de manière quelconque). Dans le cas de la règle de la *première descente*, l'algorithme choisit la première solution rencontrée qui est strictement meilleure que la solution courante.

En général, plus un voisinage est grand, plus il est difficile, en terme de temps de calcul, de trouver des optima locaux, mais plus il est probable que ces optima locaux soient de bonne qualité (si on utilise la règle de plus profonde descente). Le voisinage le plus grand qui soit est un voisinage comprenant l'ensemble des solutions du problème. Dans ce voisinage tout optimum local est donc un optimum global. Mais dans ce cas là, si le problème est NP-difficile, il n'est pas possible (si $P \neq NP$) de trouver en un temps polynomial la meilleure solution du voisinage. Il est donc nécessaire d'avoir des voisinages grands, mais explorables en un temps polynomial. Dans un voisinage de taille polynomiale (en la taille de l'instance) il est possible de regarder une à une toutes les solutions voisines et de choisir la meilleure (ou une quelconque en cas d'égalité). Un *voisinage exponentiel* (ou voisinage de taille exponentielle) est un voisinage dans lequel le nombre de solutions voisines d'une solution donnée peut être exponentiel (en la taille de l'instance). Dans un voisinage de taille exponentiel il n'est donc pas possible d'explorer (en un temps polynomial) le voisinage en énumérant toutes les solutions et en sélectionnant la meilleure. Cependant, certains voisinages de taille exponentielle peuvent être explorés en un temps polynomial en utilisant un algorithme adéquat. En utilisant ces voisinages nous pouvons alors trouver la meilleure solution parmi un très grand nombre d'autres solutions voisines, et il est donc probable que cette solution soit de bonne qualité. Depuis quelques années de nombreux travaux consistent à étudier des voisinages de taille exponentielle et à montrer que ces voisinages peuvent (ou non) être explorés en un temps polynomial [2]. Nous nous intéressons dans le chapitre 3 à un voisinage de taille exponentielle explorable en un temps polynomial.

1.2 Théorie des jeux algorithmique

La *théorie des jeux* est un domaine, au croisement des mathématiques appliquées et de l'économie, qui étudie des situations stratégiques dans lesquels les protagonistes (appelés agents, ou joueurs) choisissent différentes actions afin de maximiser leur propre profit. À l'origine développée comme un outil pour comprendre les comportements économiques, la théorie des jeux est maintenant utilisée dans de nombreux domaines : dans le domaine militaire, en biologie, informatique, psychologie, philosophie et sciences politiques notamment. La *théorie des jeux algorithmique* a connu, avec le développement d'internet, un véritable essor depuis quelques années. En effet, la théorie des jeux algorithmique (ou “algorithmic game theory” en anglais) traite de problèmes motivés par internet et d'autres réseaux décentralisés, qui ne sont pas dirigés par une seule entité centrale omnisciente mais qui sont le fruit d'interactions entre des protagonistes aux intérêts divergents. Les principales questions étudiées sont l'analyse des performances d'un tel réseau (voir section 1.2.3) et la conception de réseaux ayant de bonnes performances (voir notamment les sections 1.2.4, 1.2.5 et 1.2.6). Les situations dans lesquelles plusieurs utilisateurs ayant chacun leurs propres intérêts se font face sont alors modélisées par un jeu dans lesquels les protagonistes sont les agents, chaque agent ayant un ensemble d'actions possibles parmi lesquelles il choisira d'effectuer une action en fonction d'une fonction objectif qui lui est propre.

Nous nous intéresserons dans ce document aux jeux non coopératifs, dans lesquels les agents ne coopèrent pas entre eux. Le but de chaque agent étant de maximiser son propre profit, sans se soucier des conséquences que cela peut avoir pour les autres agents, nous appelerons dans ce document de tels agents, des *agents individualistes*. Nous dirons d'agents dont la stratégie est de choisir la ressource sur laquelle se placer (une machine dans un problème d'ordonnancement : voir chapitre 5, ou un lien dans un réseau : voir chapitre 7) qu'ils sont *autonomes*. Nous donnons ici brièvement la définition d'un jeu non coopératif (pour plus de détails voir le livre de référence en théorie des jeux [65]).

Un jeu non coopératif est constitué d'un ensemble d'agents N, chacun ayant un ensemble fini de stratégies (actions) possibles. L'ensemble des stratégies à la disposition de l'agent $i \in N$ est noté A_i. Chaque agent i doit choisir une stratégie $a_i \in A_i$ (il s'agit alors d'une stratégie *pure*). Soit $a = (a_1, \ldots, a_N)$ l'ensemble des stratégies choisies par les agents. Soit $i \in N$. Nous notons a_{-i} l'ensemble des stratégies choisies par tous les agents excepté l'agent i. Étant donné a_{-i} et une stratégie $a_i \in A_i$, nous notons (a_{-i}, a_i) l'ensemble des stratégies choisies par les agents $((a_{-i}, a_i) = a)$. Chaque agent $i \in N$ possède une fonction objectif p_i qu'il cherche à maximiser. La valeur $p_i(a)$ indique le profit de l'agent i dans le cas où les stratégies des agents sont a.

Généralement, la stratégie de chaque agent est connue par les autres agents, chaque agent choisissant sa stratégie en fonction des stratégies choisies par les autres agents (la section 1.2.2 définie alors une situation d'équilibre). Il est également possible, dans certains contextes, qu'un agent ne choisisse non pas une seule stratégie mais un ensemble de stratégies potentielles. On dit qu'un agent a une stratégie *mixte* s'il ne choisit non plus une seule stratégie de A_i mais un ensemble de stratégies $\mathcal{S}_i = \{a_{i,1}, \ldots, a_{i,|A_i|}\} \subseteq A_i$, chacune d'elle étant associée à une probabilité $prob_{i,1}, \ldots, prob_{i,|A_i|}$ (telles que $\sum_{j=1}^{|A_i|} prob_{i,j} = 1$). Les autres agents ne connaissent donc pas la stratégie finale de i mais la distribution de probabilités correspondant à la stratégie finale de i, la stratégie finale de l'agent étant en effet choisie aléatoirement parmi les stratégies de $\mathcal{S}$ avec les probabilités $prob$ associées à chaque stratégie. La section 1.2.2 présente la notion d'équilibre de Nash, primordiale pour décrire des situations d'équilibres dans des jeux non-coopératifs. Nous allons tout d'abord présenter deux modèles d'ordonnancements de tâches individualistes : nous utiliserons ces modèles pour illustrer les notions introduites dans la suite de ce chapitre. Ces modèles seront également utilisés dans les chapitres 5 et 6.

1.2.1 Ordonnancement de tâches individualistes : deux modèles

Nous nous intéressons dans cette section à l'ordonnancement de tâches indépendantes et individualistes, et nous présentons les deux principaux modèles qui modélisent le coût d'une tâche dans un ordonnancement. Le but d'une tâche individualiste étant de réduire son coût, ces modèles présentent donc les deux fonctions objectifs les plus courantes pour de telles tâches. Nous présentons maintenant ces deux modèles : les modèles KP et CKN, dont les noms sont les acronymes des noms des auteurs qui les ont fait connaître. Dans ces modèles, nous avons n agents rationnels (tâches) $1, \ldots, n$ qui doivent être exécutés sur m machines $P_1, \ldots, P_m$. Chaque tâche i est caractérisée par sa longueur (sa durée d'exécution) l_i et par son numéro d'identification i, et chaque machine P_j est caractérisée par son numéro d'identification j.

Chaque agent a une liberté d'action, et va, selon cette liberté d'action, utiliser une stratégie qui minimise son coût (voir plus bas pour la définition des coûts dans chacun des modèles). Comme nous l'avons vu dans l'introduction, la liberté d'action d'une tâche peut par exemple consister à choisir la machine sur laquelle elle va être exécutée. La stratégie d'un agent est alors soit *pure*, i.e. cet agent décide d'être exécuté sur une machine particulière (avec une probabilité égale à un), ou *mixte*, i.e. sa stratégie est une distribution de probabilités sur les machines (elle va sur chaque machine avec une probabilité donnée). La liberté d'action d'une tâche peut également être de déclarer sa longueur. Dans ce cas, la stratégie d'un agent est généralement pure : une tâche déclare une seule valeur comme étant sa longueur.

Le choix de la stratégie d'une tâche est basé sur le *coût* qui est associé à chaque stratégie. Étant données les stratégies des autres tâches, chaque tâche adopte une stratégie qui minimise son coût.

Soit c_i^j le coût de l'agent i s'il est exécuté sur la machine j. Dans chacun des modèles, le *coût* de l'agent i sera alors :

$$c_i = \min_j c_i^j.$$

Dans le premier modèle, introduit par E. Koutsoupias et C. Papadimitriou dans [55] et appelé le **modèle KP**, le but de chaque tâche $i \in \{1, \ldots, n\}$, est de minimiser la charge de la machine sur laquelle elle est exécutée (i.e. elle souhaite minimiser la somme des longueurs des tâches ordonnancées sur la même machine qu'elle). Soit p_k^j la probabilité que la tâche k soit sur la machine j, l'espérance du coût de l'agent i sur la machine j est alors l'espérance de la date de fin de la machine j, c'est à dire :

$$c_i^j = l_i + \sum_{k \neq i} p_k^j \, l_k.$$

Dans le second modèle, introduit par G. Christodoulou, E. Koutsoupias et A. Nanavati dans [30], et appelé le **modèle CKN**, chaque machine a une politique (ou algorithme local) publique qui détermine l'ordre dans lequel les tâches affectées à cette machine seront ordonnancées. Cette politique peut être une fonction de la longueur des tâches, de leur numéro d'identification, etc. Les tâches connaissent les politiques des machines, et le but de chaque tâche est de minimiser sa date de fin. L'espérance du coût de l'agent i affecté à la machine j est alors :

$$c_i^j = l_i + \sum_{k \prec_j i} p_k^j \, l_k,$$

où $k \prec_j i$ signifie que la tâche k est ordonnancée avant la tâche i selon la polique de la machine j.

Dans le modèle KP, les tâches sont exécutées selon la politique *round and robin* : elles sont découpées en plusieurs petites parties et ordonnancées les unes après les autres. La date de fin d'une tâche est alors assimilée à la date de fin de l'ordonnancement. Ce modèle a été introduit dans l'article [55] , qui peut être considéré comme l'un des articles fondateurs de la théorie des jeux algorithmique et de nombreux travaux se basent sur ce modèle. Au contraire, dans le modèle CKN, les tâches sont exécutées en un seul bloc. Il existe moins de travaux qui étudient ce modèle dans le cadre de la théorie des jeux algorithmique [30, 51]. Dans les chapitres 5 et 6 nous étudierons ce modèle sous l'angle de la stabilité d'un ordonnancement (chapitre 5), et sous l'angle de la véracité garantie d'un algorithme (chapitre 6).

1.2.2 Équilibre de Nash

Un équilibre de Nash est une solution dans laquelle aucun agent n'a intérêt à changer de stratégie, compte tenu des stratégies des autres agents. Si chaque agent a choisi une stratégie, et si aucun agent ne peut améliorer son profit en changeant de stratégie alors que les stratégies des autres agents ne changent pas, alors l'ensemble courant de stratégies constitue un équilibre de Nash. Plus formellement, un *équilibre de Nash* (pur) a^* est une situation dans laquelle, pour chaque agent $i \in N$, $p_i(a^*_{-i}, a^*_i) \geq p_i(a^*_{-i}, a_i)$, pour chaque stratégie pure $a_i \in A_i$. Il s'agit donc d'une situation dans laquelle aucun agent n'a intérêt à changer unilatéralement de stratégie.

Nous parlons ici d'équilibres de Nash purs, dans lequels la stratégie de chaque agent est pure. Si les agents ont des stratégies mixtes, alors la situation d'équilibre est décrite par un équilibre de Nash mixte. Comme les équilibres de Nash purs, un équilibre de Nash mixte est une situation dans laquelle aucun agent n'a intérêt à changer unilatéralement de stratégie : étant données les stratégies des autres agents, aucun agent ne peut augmenter son espérance de profit en changeant de stratégie.

L'ensemble des équilibres de Nash mixtes est donc plus vaste que celui des équilibres de Nash purs, et l'ensemble des équilibres de Nash mixtes comprend en particulier l'ensemble des équilibres de Nash purs (dans lequels, pour chaque $i \in N$, toutes les probabilités associées aux stratégies de $\mathcal{S}_i$ sont égales à 0, sauf une qui est égale à 1). Un équilibre de Nash pur n'existe pas toujours, mais un équilibre de Nash mixte existe toujours (quel que soit le jeu considéré), comme l'a montré J. Nash dans [63].

Considérons par exemple la situation suivante : n tâches, ou agents, souhaitent être ordonnancées sur m machines. Ces tâches sont autonomes : elles choisissent elles-même la machine sur laquelle elles seront ordonnancées. Une stratégie pure consiste donc à choisir une machine où aller, tandis qu'une stratégie mixte consiste à choisir différentes machines, chaque choix étant associé à une probabilité. Nous supposons que le but de chaque tâche est de minimiser la charge de sa machine, i.e. la somme des longueurs des tâches sur sa machines (nous nous plaçons donc dans le modèle KP).

FIG. 1.2 – *Gauche :* Équilibre de Nash. *Droite :* Solution n'étant pas un équilibre de Nash.

La figure 1.3 à gauche montre un ordonnancement qui est un équilibre de Nash pur, tandis que la figure 1.3 à droite représente une solution qui n'est pas un équilibre de Nash car la tâche de longueur 1 diminuerait la charge de sa machine en allant sur P_2 (la charge actuelle de sa machine est égale à 5, alors que si cette tâche allait sur P_2 la charge de sa machine serait égale à 3).
La solution dans laquelle toutes les tâches décident d'aller avec une probabilité $\frac{1}{m}$ sur chacune des

m machines est un équilibre de Nash mixte car aucune tâche ne peut diminuer la charge de sa machine est changeant unilatéralement de stratégie.

Notons également que le concept d'équilibre de Nash n'est pas uniquement dû à J. Nash puisque par exemple A. Cournot a montré comment trouver ce qui est maintenant appelé un équilibre de Nash dans un certain jeu (le problème du duopole). Certains auteurs se réfèrent ainsi à un équilibre de Nash-Cournot.

1.2.3 Prix de l'anarchie

Le *prix de l'anarchie* (quelquefois aussi appelé *rapport de coordination*) a été introduit par E. Koutsoupias et C. Papadimitriou dans [55] et [67], et étudié dans de nombreux problèmes par la suite. Il consiste à mesurer la perte de performance (vis-à-vis d'une fonction objectif globale) d'un système dans lequel les ressources sont allouées à des agents individualistes. Le prix de l'anarchie est en effet égal au rapport entre la valeur de la fonction objectif globale dans le pire équilibre de Nash, divisée par la valeur de la fonction objectif globale dans une solution optimale pour cette fonction objectif. Ainsi nous pouvons voir le prix de l'anarchie (ou rapport de coordination) comme la mesure qui capture le manque de coordination entre des agents indépendants et individualistes, par opposition au rapport d'approximation (qui mesure le manque de ressources de calcul), et le rapport de compétitivité (qui mesure le manque de connaissances du futur).

Définissons maintenant plus formellement le prix de l'anarchie dans le cas où les stratégies des agents sont pures. Soit un problème Π constitué d'un ensemble d'instances D_Π. Chaque instance I est constituée de données et de $N(I)$ agents ayant chacun un ensemble de caractéristiques, un ensemble de stratégies possibles, et une fonction objectif individuelle. Soit $\mathcal{E}(I)$ l'ensemble des solutions possibles (i.e. l'ensemble des situations possibles dans lesquelles chaque agent i adopte une stratégie parmi les stratégies qu'il à sa disposition dans A_i). Soit une fonction objectif globale f qui associe à chaque solution possible une valeur réelle. Nous supposons que nous souhaitons minimiser cette fonction objectif globale. Soit $\mathcal{N}(I)$ l'ensemble des équilibres de Nash (purs) de l'instance I ($\mathcal{N}(I)$ est donc l'ensemble des solutions dans lesquelles aucun agent ne peut améliorer le coût de sa fonction objectif individuelle en changeant de stratégie). Soit $OPT(I)$ le coût minimal d'une solution possible de l'instance I (cette solution n'est donc pas nécessairement un équilibre de Nash) : $OPT(I) = \min_{s\in\mathcal{E}(I)} f(s)$. Le prix de l'anarchie est égal à la valeur de la fonction objectif globale dans le pire équilibre de Nash, divisée par la valeur de la fonction objectif dans une solution optimale. Le prix de l'anarchie est défini s'il existe au moins un équilibre de Nash pour chaque instance. Si $\forall I \in D_\Pi, \mathcal{N}(I) \neq \emptyset$, alors le prix de l'anarchie est le suivant :

$$\text{Prix de l'anarchie} = \max_{I\in D_\Pi,\ e\in\mathcal{N}(I)} \frac{f(e)}{OPT(I)}$$

Si les stratégies des agents sont mixtes, alors le prix de l'anarchie est aussi égal à la valeur de la fonction objectif globale dans le pire équilibre de Nash (mixte), divisée par la valeur de la fonction objectif dans une solution optimale.

Considérons par exemple le problème d'ordonnancement cité dans la section précédente : les tâches (agents) cherchent à minimiser la charge (ou date de fin) de la machine sur laquelle elles se trouvent, et elles ont la possibilité de choisir elles-même la machine sur laquelle elles seront ordonnancées. La fonction objectif globale est la minimisation de la date de fin de l'ordonnancement. Le prix de l'anarchie est égal à $\Theta\left(\frac{\log m}{\log\log m}\right)$ [34, 54]. Le pire équilibre de Nash (mixte) est celui dans lequel tous les agents choisissent avec une probabilité $\frac{1}{m}$ d'aller sur chaque machine.

Nous présentons dans la section suivante les *mécanismes de coordination*, destinés à diminuer le prix de l'anarchie dans un réseau donné.

1.2.4 Mécanisme de coordination

Les *mécanismes de coordination* ont été introduits par G. Christodoulou, E. Koutsoupias et A. Nanavati dans [30]. Les auteurs étudient pour cela le problème suivant : un ensemble de n tâches de différentes longueurs doivent se déplacer d'une source commune à une destination commune à travers un ensemble de m liens parallèles identiques. Ce problème peut être vu comme un problème d'ordonnancement dans lequel n tâches doivent être routées sur m machines identiques. Chaque tâche est caractérisée par une longueur et un numéro d'identification, et est détenue par un agent indépendant et autonome dont le but est de minimiser la date de fin de sa tâche. Chaque machine a une *politique* qui lui permet de donner un ordre aux tâches qu'elle doit ordonnancer. Chaque machine ne connaît que les tâches qu'elle doit ordonnancer et la politique d'une machine ne doit donc pas dépendre des tâches routées sur les autres machines. La politique d'une machine détermine l'ordre dans lequel ordonnancer ses tâches et peut également éventuellement introduire des temps d'inactivité entre les tâches (i.e. une machine peut, pendant un certain temps, rester inactive alors que certaines tâches n'ont pas encore été ordonnancées). Connaissant les caractéristiques des autres tâches et la politique des machines, chaque tâche choisit sur quelle machine être ordonnancée. L'ensemble des politiques des machines est appelé un *mécanisme de coordination*. Un mécanisme de coordination est intéressant s'il engendre des équilibres de Nash de bonne qualité vis-à-vis de la fonction objectif globale que l'on souhaite optimiser. En effet, chaque tâche se comportant de façon à minimiser sa date de fin, la solution obtenue sera un équilibre de Nash. Souvent, un mécanisme de coordination engendre un seul équilibre de Nash pur, dans lesquel chaque agent peut choisir sa stratégie uniquement en fonction des caractéristiques des autres agents : les agents convergent ainsi très rapidement vers un équilibre de Nash.

À titre d'exemple, considérons le mécanisme de coordination suivant, donné dans [30], et qui est valable dans le cas où il y a deux machines : la première machine, nommée P_{SPT} ordonnance ses tâches de la plus petite à la plus grande, et la deuxième machine, nomée P_{LPT} ordonnance ses tâches de la plus grande à la plus petite (si deux tâches ont la même longueur, alors la plus grande est celle qui a le plus grand numéro d'identification). Il est facile de voir qu'en présence d'un tel mécanisme de coordination, la plus petite tâche ira sur P_{SPT}, et la plus grande tâche sur P_{LPT}. Plus généralement, une tâche i ira sur P_{SPT} seulement si la somme des longueurs des tâches plus petites que i est inférieure ou égale à la somme des longueurs des tâches plus grandes que i.

Avec ce mécanisme de coordination, l'ordonnancement le plus mauvais obtenu vis-à-vis de la date de fin de l'ordonnancement a un rapport d'approximation égal à $\frac{4}{3}$: le rapport de coordination de ce mécanisme de coordination est donc de $\frac{4}{3}$. Ce mécanisme de coordination n'est pas le meilleur, puisque les auteurs de [30] donnent un mécanisme de coordination qui a un rapport de coordination de $\frac{4}{3} - \frac{1}{3\,m}$, et conjecturent qu'aucun mécanisme de coordination ne peut avoir un meilleur rapport. Dans ce dernier mécanisme de coordination, chaque machine ordonnance ses tâches par ordre de longueurs décroissantes, et ajoute entre les tâches des petits délais afin qu'il y ait pour chaque tâche uniquement une machine sur laquelle elle puisse minimiser sa date de fin. Notons cependant que ce mécanisme de coordination suppose que chaque machine connaisse une valeur ε qui soit plus petite que la longueur de chaque tâche (ceci est donc légèrement moins général qu'un mécanisme de coordination ne faisant aucune hypothèse sur la longueur des tâches). Nous remarquons que ces mécanimes de coordination font baisser le prix de l'anarchie du réseau, puisque si les machines ordonnancent les tâches selon un ordre aléatoire par exemple, alors la situation dans laquelle chaque tâche a une stratégie mixte qui consiste à aller sur chaque machine avec une même probabilité serait un équilibre de Nash mixte, et le prix de l'anarchie ne serait alors pas constant mais égal à $\Theta\left(\frac{\log m}{\log\log m}\right)$ [34, 54].

Le but d'un mécanisme de coordination est donc d'induire un prix de l'anarchie le plus petit possible. Nous avons jusqu'à présent présenté les mécanismes de coordination pour un problème d'ordonnancement uniquement. Ceci peut être facilement généralisé au cas où des agents indépendants et autonomes souhaitent occuper des ressources communes (dans le cas ci-dessus les agents sont les tâches et les ressources les machines, mais les ressources peuvent par exemple être les liens d'un réseau, ou tout autre ressource partagée entre plusieurs agents). Il suffit pour cela que chaque ressource ait une *politique* publique (connue des agents) qui est un algorithme local qui lui indique comment gérer les différentes demandes des agents. Nous sommes dans un contexte totalement distribué : chaque ressource ne connaît donc que les caractéristiques des agents ayant demandé à l'utiliser, et sa politique ne doit donc dépendre que de ces caractéristiques (et non pas des caractéristiques des autres agents).

Notons que nous utiliserons des mécanismes de coordination dans les chapitres 6 (dans le cadre d'un problème d'ordonnancement) et 7 (dans le cadre d'un problème de routage de paquets). Pour des raisons de facilité de lecture, nous utiliserons quelquefois l'expression "prix de l'anarchie d'un mécanisme de coordination" pour désigner le prix de l'anarchie obtenu *avec* ce mécanisme de coordination.

1.2.5 Prix de la stabilité

Le *prix de la stabilité* a été introduit par A. Schultz et N. Stier Moses dans [73], puis par E. Anshelevich et al. dans [17] qui l'ont nommé ainsi. Le prix de la stabilité évalue l'impact (sur une fonction objectif donnée) de ne chercher que des solution stables (des équilibres de Nash). Son utilisation est justifiée par l'observation suivante : dans de nombreuses applications, les utilisateurs (ou agents) n'intéragissent pas directement les uns avec les autres, mais font face à un protocole qui propose une solution collective aux agents, qui sont ensuite libres d'accepter ou non cette solution. Dans ces applications, il est alors nécessaire de produire des solutions stables (i.e. telles qu'aucun agent ne refuse la solution) et de la meilleure qualité possible vis-à-vis de la fonction objectif considérée. Ces solutions correspondent donc aux meilleurs équilibres de Nash vis-à-vis de cette fonction objectif.

Le prix de la stabilité est égal à la valeur de la fonction objectif globale dans le meilleur équilibre de Nash pur, divisée par la valeur de la fonction objectif dans une solution optimale. Nous utilisons les mêmes notations que dans la section 1.2.3 : D_Π est l'ensemble des instances possibles, $\mathcal{N}(I)$ l'ensemble des équilibres de Nash purs de l'instance I, f est la fonction objectif globale considérée, et $OPT(I)$ est le coût de la meilleure solution possible de l'instance I (vis-à-vis de la fonction objectif f). Le prix de la stabilité est défini s'il existe au moins un équilibre de Nash pur pour chaque instance. Si $\forall I \in D_\Pi, \mathcal{N}(I) \neq \emptyset$, alors le prix de la stabilité est le suivant :

$$\text{Prix de la stabilité} = \max_{I \in D_\Pi} \min_{e \in \mathcal{N}(I)} \frac{f(e)}{OPT(I)}$$

Par exemple, supposons que nous avons n tâches à ordonnancer sur deux machines. Le but de chaque tâche est de minimiser la date de fin de sa machine (nous sommes donc dans le modèle KP défini dans la section 1.2.1). Le prix de la stabilité est alors égal à un, car un ordonnancement optimal est un équilibre de Nash : dans un tel ordonnancement les tâches de la machine la moins chargée n'ont clairement pas intérêt à changer de machine, et les tâches de la machine la plus chargée non plus (si une tâche de la machine la plus chargée diminuait la date de fin de sa machine en changeant de machine, alors cet ordonnancement ne serait pas optimal). Notons que les équilibres de Nash dans ce contexte ne sont pas tous optimaux. Par exemple, la solution de la figure 1.3 à droite est un équilibre de Nash non optimal : le prix de la stabilité est donc ici différent du prix de l'anarchie.

FIG. 1.3 – *Gauche :* Équilibre de Nash optimal. *Droite :* Équilibre de Nash non optimal.

1.2.6 Algorithmes avec véracité garantie

Un aspect important des protocoles (algorithmes) qui font face à des agents individualistes est la notion de *véracité* des informations détenues par le protocole. Considérons un système avec des agents individualistes (dont le but est de maximiser leur propre fonction objectif), et géré par un protocole dont le but est de maximiser une fonction objectif globale. L'hypothèse habituellement faite est que les agents qui agissent avec le protocole sont dignes de confiance. Cependant, dans certains cas, il n'est pas réaliste de faire une telle hypothèse car les agents peuvent essayer de manipuler le protocole à leur avantage en déclarant de fausses informations. Avec de fausses informations, même le protocole le plus performant peut mener à des solutions non raisonnables s'il n'est pas construit de manière à tenir compte des comportements individualistes des agents. Ainsi, si chaque agent possède une valeur (par exemple, pour une tâche sa longueur, pour une machine sa vitesse, etc.) qui est connue uniquement de lui-même, et si le protocole a besoin de connaître cette valeur, alors il est utile de concevoir des protocoles dans lesquels les agents ont intérêt (vis-à-vis de leur propre fonction objectif) à déclarer leur vraies valeurs. De tels protocoles (algorithmes) sont dits avec (ou à) *véracité garantie* ("truthful" en anglais). Lorsqu'un algorithme avec un rapport d'approximation ρ est à véracité garantie, nous pouvons garantir que la solution qu'il retourne est réellement ρ-approchée, ce qui n'est pas forcément le cas d'un algorithme ρ-approché mais qui n'est pas à véracité garantie.

Le domaine de la conception d'algorithmes à véracité garantie ("(algorithmic) mechanism design" en anglais) est assez récent en informatique théorique, alors qu'il est utilisé depuis de nombreuses années en économie notamment. Dans la plupart des cas, ces mécanismes requièrent le paiement des agents, afin de les inciter à déclarer leurs valeurs privées réelles. Ceci est possible si le coût de chaque agent (la valeur de sa propre fonction objectif) peut s'exprimer de façon monétaire. Il est alors courant de payer les agents et de supposer que chaque agent cherche à maximiser son profit global, i.e. la somme du paiement reçu et du coût induit par la solution retournée par le mécanisme. Un mécanisme $M = (A, \mathcal{P})$ est donc un couple où A est un algorithme qui retourne une solution du problème et $\mathcal{P}$ est une fonction de paiement qui affecte à chaque agent une certaine quantité d'argent. Un mécanisme est alors à véracité garantie si et seulement si le profit de chaque agent est maximisé quand l'agent révèle sa vraie valeur.

La technique la plus célèbre pour obtenir des mécanismes avec véracité garantie est probablement le mécanisme VCG, dont le nom vient de ses concepteurs, Vickrey, Clarke et Groves [80, 31, 43]. Ce mécanisme est avec véracité garantie quelque soit la fonction objectif des agents, mais sous l'hypothèse que la fonction objectif globale est égale à la somme des coûts des agents, et que le mécanisme est capable de calculer la solution optimale. Ce mécanisme est souvent utilisé dans des problèmes avec enchères, et a également commencé à être utilisé en informatique théorique. Il peut par exemple être utilisé si les agents sont des tâches qui souhaitent être ordonnancées sur des machines parallèles et dont le but est de réduire leur coût, qui est proportionnel à la date à laquelle elles se terminent (i.e. les tâches souhaitent être exécutée le plus rapidement possible), et si la fonction objectif globale est de réduire la date de fin moyenne des tâches (cette fonction objectif est en effet équivalente à minimiser la somme des dates de fin de l'ensemble des tâches). Par contre quand un problème est NP-difficile alors ce mécanisme peut difficilement être utilisé car il est nécessaire de trouver la solution optimale du problème (ce qui ne peut pas être fait en un temps polynomial si $P \neq NP$).

Une autre technique importante a été introduite par A. Archer et É. Tardos [18] pour les problèmes du type suivant : un ensemble de charges doivent être allouées à un ensemble de ressources (qui sont les agents) et chaque agent possède une valeur privée qui est un nombre positif réel représentant le coût qu'une unité de charge induit à l'agent. Les auteurs de [18] montrent que pour ces problèmes un mécanisme $M = (A, \mathcal{P})$ est avec véracité garantie si et seulement si l'algorithme A est *monotone* : un algorithme est monotone si, étant données les valeurs $b_1, \ldots, b_n$ déclarées par les agents, alors pour chaque i (les valeurs b_j étant fixées pour $j = 1, \ldots, n$ et $j \neq i$), la charge de l'agent i ne diminue pas quand b_i diminue. De plus les auteurs donnent une fonction de paiement $\mathcal{P}_A$ telle que si A est monotone alors $(A, \mathcal{P}_A)$ est un mécanisme avec véracité garantie. Cette technique peut par exemple être utilisée dans le problème d'ordonnancement suivant : les agents sont des machines parallèles ayant chacune une vitesse d'exécution, les charges sont des tâches à ordonnancer, et la valeur secrète de chaque machine est sa vitesse, ou plus exactement l'inverse de sa vitesse : le coût (temps) que met la machine à exécuter une unité de tâche. Dans cet exemple, la monotonicité signifie qu'un algorithme est avec véracité garantie seulement si, quand on augmente la vitesse d'une seule machine alors l'algorithme ne diminue pas la quantité de tâches affectées à cette machine.

Dans le chapitre 6 nous étudierons des mécanismes avec véracité garantie pour le problème qui consiste à ordonnancer, sur des machines parallèles, des tâches (agents) dont le but est de minimiser leur date de fin, alors que l'objectif global sera de minimiser la date de fin de l'ordonnancement.

Chapitre 2

Groupage de trafic dans un réseau optique WDM en étoile

Dans ce chapitre, nous nous intéressons au routage de paquets (ou demandes de trafic) dans un réseau optique qui a la forme d'une étoile. Plusieurs longueurs d'ondes sont disponibles pour router les paquets entre deux sommets, et il est possible de grouper plusieurs paquets dans une même longueur d'onde, mais ceci engendre un coût important en terme de stockage, calcul, et temps de parcours de ces paquets. Notre but est donc d'optimiser l'agencement des paquets afin de minimiser la quantité de trafic qui doit partager une longueur d'onde. Ces travaux ont été présentés dans [7].

2.1 Introduction

Les réseaux optiques (constitués de fibres optiques) possèdent de nombreux avantages : ils sont adaptés aux communications longues en terme de distance physique, ils possèdent de très hauts débits, ils n'émettent pas de radiations, sont difficilement piratables (il est impossible de couper une fibre optique sans la détruire), et possèdent un très faible taux d'erreur (nombre de paquets perdus) par rapport aux autres réseaux classiques.

Une fibre optique est un fil transparent très fin qui conduit la lumière. Entourée d'une gaine protectrice, la fibre optique peut être utilisée pour conduire la lumière entre deux lieux potentiellement très éloignés. Le signal lumineux codé par une variation d'intensité est capable de transmettre une grande quantité d'informations. Plusieurs longueurs d'ondes sont généralement présentes dans une fibre optique et, en utilisant la technologie WDM (pour "Wavelengh Division Multiplexing"), chacune de ces longueurs d'ondes peut coder un paquet. Afin d'utiliser plus efficacement la bande passante d'un réseau optique, il est possible, depuis quelques années, de grouper plusieurs paquets dans une même longueur d'onde. En effet, puisqu'il y a en général plus de paquets à router que de longueurs d'ondes disponibles, il est intéressant de grouper certains paquets dans une même longueur d'ondes, afin que tous les paquets puissent être routés.

Nous nous intéressons aux réseaux en forme d'étoile, composés d'un ensemble d'émetteurs, d'un ensemble de récepteurs, et d'un commutateur au centre. Les paquets qui ne possèdent pas une longueur d'onde à eux seuls de leur source à leur destination doivent alors être traités électroniquement au commutateur : le signal de lumière doit être converti sous forme électronique, stocké électroniquement dans un buffer, traité (i.e. les paquets groupés entrant sont séparés, certains sortant sont groupés sur la même longueur d'onde), puis converti à nouveau sous forme de lumière. Ceci présente plusieurs inconvénients : il faut tout d'abord stocker les requêtes électroniquement, ce qui peut nécessiter une quantité très importante de mémoire ; il faut aussi avoir une puissance de calcul pour effectuer les conversions entre le signal lumineux et les données électroniques, ainsi que le traitement de ces données électroniques ; enfin ces conversions et calculs demandent un temps important (le temps de routage est multiplié par plus de 100 quand il est nécessaire de transformer le signal lumineux sous forme électronique puis à nouveau sous forme de lumière). Pour ces raisons, notre but est de minimiser la quantité de trafic traité électroniquement par le commutateur. Notre intérêt pour les réseaux en étoile, est d'une part motivée par leur simplicité, qui nous permet de donner les premiers algorithmes avec garantie de performance pour ce problème de groupage de trafic, et d'autre part par leur intérêt pratique. En effet, ces réseaux sont souvent utilisés dans l'interconnexion de réseaux locaux (LAN, pour "local area network") ou de moyennes distances (MAN, pour "metropolitan area network") avec l'épine dorsale ("backbone") d'un réseau étendu (WAN, pour "wide area network").

Présentation du problème

Nous considérons un réseau dont le graphe sous-jacent est une étoile ayant $N+1$ sommets. Il y a donc un seul sommet, appelé *commutateur*, qui est relié aux autres sommets par un lien physique. Tous les sommets, à l'exception du commutateur, sont partitionnés en deux groupes V_1 et V_2 : les sommets de V_1 sont les émetteurs, et les sommets de V_2 sont les récepteurs. Le commutateur reçoit le numéro 0, et les N autres sommets reçoivent un numéro entre 1 et N dans un ordre quelconque. Chaque lien physique consiste en une fibre optique, et chaque fibre porte W longueurs d'ondes. Chaque longueur d'onde a une capacité C (exprimée en unités d'une mesure quelconque). Le trafic entre les sommets est modélisé sous la forme d'une matrice de trafic $T = [t_{ij}]$, où l'entier t_{ij} est égal au nombre d'unités de trafic que le sommet $i \in V_1$ souhaite envoyer au sommet $j \in V_2$. Nous utiliserons aussi dans la suite le symbole t_{ij} pour représenter la requête, ou demande de trafic, du sommet i vers le sommet j. Nous n'autorisons pas les demandes de trafic à être supérieures à la capacité d'une longueur d'onde, i.e. pour tout $(i, j), 0 \leq t_{ij} \leq C$.

Nous remarquons qu'en considérant un ensemble d'émetteurs V_1 et un ensemble de récepteurs V_2, nous pouvons aussi représenter un réseau en étoile dans lequel chaque sommet est à la fois un émetteur et un récepteur, et dans lequel il y a, entre chaque noeud et le commutateur, une fibre optique dans chaque direction (chaque fibre porte W longueurs d'ondes) : une telle instance peut en effet être transformée en une instance dans laquelle il y a une fibre optique de chaque émetteur vers le commutateur, et une fibre du commutateur vers chaque récepteur. Il suffit pour cela de remplacer chaque sommet i par un émetteur i_1 et un récepteur i_2, et chaque demande de trafic t_{ij} par t_{i_1,j_2}.

Le commutateur possède des capacités optiques et électroniques : il peut laisser certaines requêtes passer de manière transparente à travers lui (il fait juste suivre la requête qui arrive vers son destinataire), mais il peut également arrêter certaines requêtes, les traiter, puis les envoyer vers leur destinataire. Une requête t_{ij} doit bénéficier d'une longueur d'onde à elle seule de i au commutateur, et du commutateur à j pour être *routée optiquement*, alors qu'elle peut partager une longueur d'onde avec d'autres requêtes si elle est commutée électroniquement. Nous dirons dans ce dernier cas qu'une requête est *routée électroniquement*. Le commutateur doit alors séparer plusieurs requêtes groupées dans la même longueur d'onde, et router ces requêtes vers leur destinataire, en les regroupant éventuellement avec d'autres requêtes. Ceci a un certain coût, de stockage des requêtes notamment, et nécessite du temps de calcul. Le but est donc de *minimiser la quantité totale de trafic commuté électroniquement*. Ce problème est communément désigné sous le terme *problème de groupage de trafic* (ou "traffic grooming problem" en anglais).

Il existe en réalité deux versions du problème de groupage de trafic : soit nous souhaitons minimiser la quantité totale de trafic routé électroniquement (il s'agit ici de la version de *minimisation du problème), soit nous souhaitons maximiser la quantité totale de trafic routé optiquement (il s'agit ici de la version de* maximisation du problème). Notons que la quantité de trafic routé électroniquement(respectivement optiquement) est égale à la somme des demandes de trafic routées électroniquement (respectivement optiquement). Ces deux versions du problème de groupage de trafic sont équivalentes (i.e. une solution optimale pour l'une est également une solution optimale pour l'autre), car le coût d'une solution optimale pour la version de maximisation du problème est égale à la somme de toutes les demandes de trafic, moins le coût d'une solution optimale de la version de minimisation du problème.

Exemple. Considérons l'instance suivante : nous avons un émetteur i, qui a des demandes de trafic vers trois récepteurs j, k, et l. Ces demandes sont les suivantes : $t_{ij} = t_{ik} = C/3$, et $t_{il} = C/2$, où C est la capacité de chaque longueur d'onde. Le nombre de longueurs d'ondes par fibre est 2 ($W = 2$). Puisqu'il n'y a que deux longueurs d'ondes à partir de i, les trois demandes de trafic ne peuvent pas être toutes routées optiquement. Si la demande de trafic t_{ij} est routée optiquement, elle occupe une longueur d'onde à elle seule entre i et le commutateur. Les demandes de trafic t_{ik} et t_{il} doivent alors partager la longueur d'onde restante vers le commutateur, et sont donc routées électroniquement : la quantité de trafic électroniquement routé est alors égale à $C/3 + C/2 = 5\,C/6$. Dans une solution optimale, t_{il} est routée optiquement, alors que t_{ij} et t_{ik} sont routées électroniquement : la quantité de trafic routé électroniquement est alors de $2\,C/3$.

Travaux connexes : De nombreux travaux portent sur des problèmes de trafic dans de nombreuses topologies (voir par exemple les articles [36, 61, 82] qui offrent une vue d'ensemble du sujet). La plupart des travaux concernent le cas dans lequel le réseau est un anneau. Dans [49], R. Dutta et G.N. Rouskas étudient le problème de groupage de trafic dans une étoile. Ils montrent que ce problème est NP-difficile, et donnent un algorithme exact (mais qui nécessite donc dans le pire des cas un temps exponentiel) pour le résoudre. Cet algorithme leur permet de donner des bornes inférieures et supérieures sur le coût d'une solution optimale du problème de groupage de trafic. Ils donnent également une heuristique gloutonne pour ce problème. Il existe cependant des différences entre leur problème et le notre : dans [49] le commutateur peut être un émetteur et un récepteur (ce qui n'est pas le cas dans notre problème car le commutateur est passif : les seuls sommets qui s'échangent des requêtes sont les feuilles de l'étoile). De plus, dans [49] les demandes de trafic entre deux sommets sont autorisées à être plus grande que la capacité d'une longueur d'onde (i.e. si i est un émetteur et j un récepteur, il est possible que $t_{ij} > C$). Pour distinguer ces deux problèmes, nous appellerons leur problème le problème de groupage de trafic dans une étoile active, alors que notre problème sera appelé le *problème de groupage de trafic dans une étoile* *passive* (voir section 2.4.2 pour une formulation de ce problème sous forme de programme linéaire en nombres entiers). Nous remarquons qu'une fois que nous savons quelles requêtes sont routées optiquement, l'affectation des requêtes aux différentes longueurs d'onde est un problème facile.

Plan du chapitre : Tout d'abord, nous montrons dans la section 2.2 que le problème de groupage de trafic dans une étoile passive est un problème NP-difficile. Nous montrons ensuite dans la section 2.3 que ce problème peut être résolu en un temps polynomial s'il y a seulement deux longueurs d'ondes par fibre ($W = 2$) : nous donnons un algorithme exact pour ce problème. Dans la section 2.4, nous nous intéressons à des algorithmes approchés pour le cas général. Nous montrons dans la section 2.4.1 que nous ne pouvons pas déduire d'algorithme d'approximation avec garantie de performance pour le problème dans lequel on souhaite minimiser la quantité de trafic routé électroniquement (i.e. pour la version de minimisation de notre problème), à partir d'un algorithme d'approximation avec garantie de performance pour le problème dans lequel on souhaite maximiser la quantité de trafic routé optiquement (i.e. pour la version de maximisation de notre problème). Ceci est également vrai dans l'autre sens : un algorithme approché avec garantie de performance pour la version de minimisation n'est pas nécessairement un algorithme approché avec garantie de performance pour la version de maximisation. Nous donnons ensuite deux algorithmes d'approximation. Le premier (section 2.4.2) concerne la version de minimisation : nous transformons notre problème en un problème de multi-couverture contrainte de multi-ensemble [70], et nous obtenons un rapport d'approximation inférieur à $\log(2\,C) + 1$. Le deuxième algorithme d'approximation (section 2.4.3) concerne la version de maximisation : nous transformons notre problème en un problème de couplage avec demandes dans un graphe biparti [74], et nous obtenons un rapport d'approximation de $(2 + \frac{4}{5})$. La section 2.5 conclut ce chapitre.

2.2 Complexité du problème

Nous montrons dans cette section que la version de décision du problème de groupage de trafic dans une étoile passive est NP-complet.

Nous nous sommes pour cela inspirés de la preuve de R. Dutta et G. Rouskas dans [49] : dans cet article, les auteurs montrent que le problème de groupage de trafic qu'ils étudient est NP-complet. Ils réduisent pour cela la version de décision du problème du sac-à-dos à leur problème. Nous faisons la même réduction, mais plusieurs adaptations sont nécessaires car notre problème est moins général que le leur. Nous remplaçons chaque demande de trafic t_{ij} supérieure à C par plusieurs demandes de trafic (dont la somme est égale à t_{ij}) qui partent de i, et par plusieurs demandes de trafic (dont la somme est égale à t_{ij}) qui vont vers j. Nous devons faire en sorte que ces demandes soient routées électroniquement. Les auteurs de [49] utilisent également dans leur réduction des demandes de trafic dont l'émetteur ou le récepteur est le commutateur. Nous remplaçons alors ces demandes de trafic par des demandes de trafic de nouveaux sommets (ou vers de nouveaux sommets) et nous forçons ces demandes de trafic à être routées électroniquement.

Nous réduisons la version du problème du sac-à-dos [40] à la version de décision de notre problème de groupage de trafic, qui est la suivante : soit une instance du problème de groupage de

trafic et soit $Q \in \mathbb{Z}^+$, existe-t-il une solution du problème de groupage de trafic dans laquelle la quantité de trafic routé optiquement est supérieure ou égale à Q ?
Une instance du problème du sac-à-dos consiste en un ensemble U contenant n éléments tels que $\forall i \in \{1, 2, ..., n\}$, l'élément $u_i \in U$ a un poids $w_i \in \mathbb{Z}^+$, et une valeur $v_i \in \mathbb{Z}^+$; un poids cible $B \in \mathbb{Z}^+$; et une valeur cible $K \in \mathbb{Z}^+$. La version de décision du problème de sac-à-dos est la suivante : existe-t-il un vecteur binaire $X = \{x_1, x_2, ..., x_n\}$ tel que $\sum_{i=1}^n x_i w_i \leq B$, et $\sum_{i=1}^n x_i v_i \geq K$. En d'autres termes, est-il possible de choisir un sous-ensemble de U tel que le poids total de ces éléments soit d'au plus B, et la valeur totale de ces éléments soit d'au moins K ?
À partir d'une instance de la version de décision du problème du sac-à-dos, nous construisons une instance de la version de décision du problème de groupage de trafic en utilisant la transformation suivante : $W = n$, $C = max_i(w_i + v_i) + 1$, $Q = K + \sum_{i=1}^n (C - w_i - v_i)$, et la matrice des demandes de trafic est représentée sur la figure 2.1. Dans cette figure, les sommets sont ceux du réseau en étoile (le commutateur n'est pas représenté), et les liens (arêtes) représentent les demandes de trafic. Les demandes de trafic égales à 0 ne sont pas représentées, et la valeur indiquée sur le lien joignant le sommet a au sommet b est $t_{a,b}$. Les sommets dont les numéros sont entre $n + 1$ et $n + 10$ représentent chacun un sommet du réseau, mais les sommets $i_\alpha, j_\alpha, k_\alpha, l_\alpha, m_\alpha, p_\alpha$ et q_α représentent chacun plusieurs sommets :
Pour les sommets i_α, α va de 1 à n (i.e. i_α représente les sommets $i_1, i_2, ..., i_n$) ;
pour les sommets j_α, α va de 1 à $\lfloor \frac{(n-2)C}{C-1} \rfloor$;
pour les sommets k_α, α va de 1 à $\lfloor \frac{\sum_{k=1}^n (w_k - B)}{C-1} \rfloor$;
pour les sommets l_α, α va de 1 à $\lfloor \frac{nC-(C-1)}{C-1} \rfloor$;
pour les sommets m_α, α va de 1 à $\lfloor \frac{nC-((\sum_{k=1}^n (w_k - B)) \bmod (C-1))}{C-1} \rfloor$;
pour les sommets p_α, α va de 1 à $\lfloor \frac{n}{C-1} \rfloor$;
pour les sommets q_α, α va de 1 à $\lfloor \frac{nC-n((n-2)C \bmod (C-1))}{C-1} \rfloor$;
et pour les sommets r_α, α va de 1 à $\lfloor \frac{nC-\sum_{k=1}^n w_k}{C-1} \rfloor$.

Lemme 2.1 *Soit a un émetteur et b un récepteur. Il n'est pas possible que la demande de trafic de a vers b soit optiquement routée, si $(a, b) \neq (n + 1, i_\alpha)$ ou $(a, b) \neq (n + 2, i_\alpha)$.*

Preuve : Montrons que chaque demande de trafic $t_{a,b}$ différente de 0 entre chaque couple (a, b) de sommets de $V_1 \times V_2$ et tels que $(a, b) \neq (n + 1, i_\alpha)$ et $(n + 2, i_\alpha)$, ne peut pas être routée optiquement. Pour cela, nous montrons que la somme des demandes de trafic à partir de a, ou la somme des demandes de trafic vers b, est égale à $n\,C$ (qui est égal à $W\,C$, la totalité de la bande passante disponible), et que $t_{a,b} < C$.

– $\forall x \in V_2, t_{n+9,x}$ ne peut pas être routée optiquement. En effet :
$$\begin{aligned}\sum_{x\in V_2} t_{n+9,x} &= \sum_\beta t_{n+9,r_\beta} + t_{n+9,n+10} + \sum_\beta t_{n+9,i_\beta}\\ &= \lfloor \tfrac{nC-\sum_{k=1}^n w_k}{C-1} \rfloor (C-1) + (nC - \textstyle\sum_{k=1}^n w_k) \bmod (C-1)\\ &\quad + \textstyle\sum_{k=1}^n w_k\\ &= nC\end{aligned}$$
et $t_{n+9,r_\alpha} < C,\ t_{n+9,n+10} < C,\ t_{n+9,i_\alpha} < C$.

– $\forall x \in V_2, t_{j_\alpha,x}$ ne peut pas être routée optiquement. En effet :
$$\sum_{x\in V_2} t_{j_\alpha,x} = \sum_\beta t_{j_\alpha,i_\beta} + \sum_\beta t_{j_\alpha,p_\beta} + t_{j_\alpha,n+7}$$

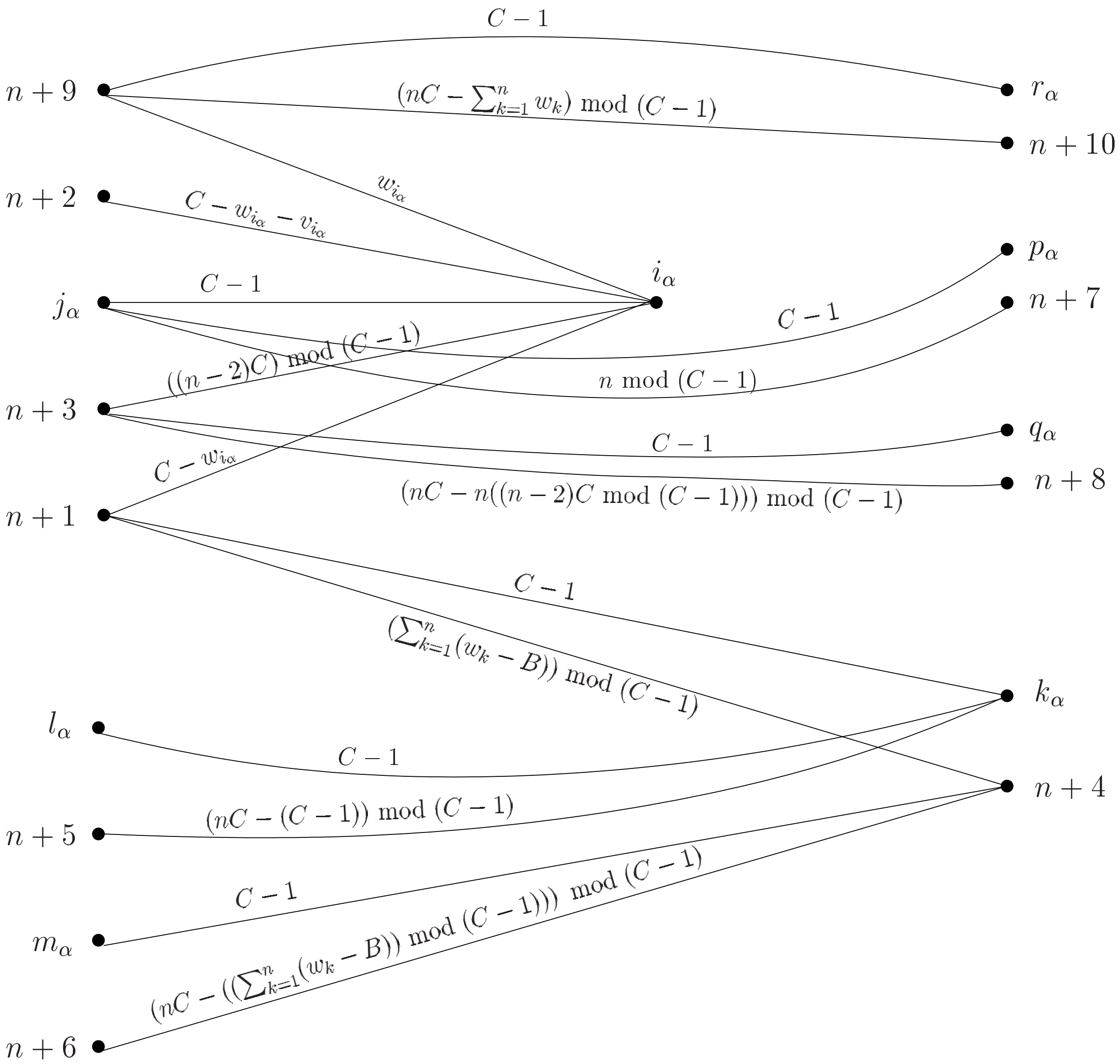


FIG. 2.1 – Illustration de la matrice des trafic. Les sommets à gauches sont les émetteurs et les sommets à droite sont les récepteurs.

$$= n(C-1) + \lfloor \tfrac{n}{C-1} \rfloor (C-1) + n \bmod (C-1)$$
$$= nC$$

et $t_{j_\alpha,i_\beta} < C,\ t_{j_\alpha,p_\beta} < C,\ t_{j_\alpha,n+7} < C$.

- $\forall x \in V_2, t_{n+3,x}$ ne peut pas être routée optiquement. En effet :
  $$\sum_{x \in V_2} t_{n+3,x} = \sum_\beta t_{n+3,i_\beta} + \sum_\beta t_{n+3,q_\beta} + t_{n+3,n+8}$$
  $$= n((n-2)C \bmod (C-1)) + \lfloor \tfrac{nC - n((n-2)C \bmod (C-1))}{C-1} \rfloor (C-1)$$
  $$+ (nC - n((n-2)C \bmod (C-1))) \bmod (C-1)$$
  $$= nC$$
  et $t_{n+3,i_\alpha} < C,\ t_{n+3,q_\alpha} < C,\ t_{n+3,n+8} < C$.
- $\forall x \in V_1, t_{x,k_\alpha}$ ne peut pas être routée optiquement. En effet :
  $$\sum_{x \in V_1} t_{x,k_\alpha} = t_{n+1,k_\alpha} + \sum_\beta t_{l_\beta,k_\alpha} + t_{n+5,k_\alpha}$$

$= (C-1) + \lfloor \frac{nC-(C-1)}{C-1} \rfloor (C-1) + (nC-(C-1)) \bmod (C-1)$
$= nC$
et $t_{n+1,k_\alpha} < C,\ t_{l_\beta,k_\alpha} < C,\ t_{n+5,k_\alpha} < C.$

– $\forall x \in V_1, t_{x,n+4}$ ne peut pas être routée optiquement. En effet :
$\sum_{x \in V_1} t_{x,n+4} = \sum_\beta t_{m_\beta,n+4} + t_{n+6,n+4} + t_{n+1,n+4}$
$= \frac{nC-((\sum_{k=1}^{n}(w_k-B)) \bmod (C-1))}{C-1}(C-1) + (\lfloor nC - ((\sum_{k=1}^{n}(w_k- B)) \bmod (C-1)) \rfloor) \bmod (C-1) + (\sum_{k=1}^{n}(w_k-B)) \bmod (C-1)$
$= nC$
et $t_{n+1,n+4} < C,\ t_{m_\beta,n+4} < C,\ t_{n+6,n+4} < C.$ □

Lemme 2.2 *Soit $\alpha \in \{1, ..., n\}$. Les demandes de trafic t_{n+1,i_α} et t_{n+2,i_α} ne peuvent pas être routées optiquement simultanément.*

Preuve : Le sommet i_α reçoit du commutateur un trafic total (sans compter t_{n+1,i_α} et t_{n+2,i_α}) égal à : $t_{n+9,i_\alpha} + \sum_\beta t_{j_\beta,i_\alpha} + t_{n+3,i_\alpha} = w_{i_\alpha} + \lfloor \frac{(n-2)C}{C-1} \rfloor (C-1) + ((n-2)C) \bmod (C-1) = (n-2)C + w_{i_\alpha} > (n-2)C$
Puisque $W = n$, il reste au plus une longueur d'onde disponible pour router une autre demande de trafic vers le sommet i_α. □

Lemme 2.3 *Soit $\alpha \in \{1, ..., n\}$. Il est possible de router optiquement une demande de trafic du sommet $n+1$ vers le sommet i_α, ou du sommet $n+2$ vers le sommet i_α.*

Preuve :

– Montrons qu'il est possible de router électroniquement la demande de trafic du sommet $n+1$ au sommet i_α :
$\sum_{x \in V_1, x \neq n+1} t_{x,i_\alpha} = (n-2)C + w_{i_\alpha} + (C - w_{i_\alpha} - v_{i_\alpha}) = (n-1)C - v_{i_\alpha} \leq (n-1)C$ et, puisque $\exists i_\alpha \in \{1, ..., n\}, B \geq w_{i_\alpha}$ (sinon l'instance du problème de sac-à-dos serait triviale) et $C = max_{i_\alpha}(v_{i_\alpha} + w_{i_\alpha}) + 1$, nous avons $\sum_{x \in V_2, x \neq i_\alpha} t_{n+1,x} = \sum_{k=1}^{n}(w_k - B) \leq (n-1)C$. Il y a donc assez de bande passante disponible pour router la demande de trafic du sommet $n+1$ au sommet i_α.
– Montrons qu'il est possible de router électroniquement la demande de trafic du sommet $n+2$ au sommet i_α :
$\sum_{x \in V_1, x \neq n+2} t_{x,i_\alpha} = (n-2)C + w_{i_\alpha} + (C - w_{i_\alpha}) = (n-1)C \leq (n-1)C$ et $\sum_{x \in V_2, x \neq i_\alpha} t_{n+2,x} = 0 \leq (n-1)C$. Il y a donc assez de bande passante disponible pour router la demande de trafic du sommet $n+2$ au sommet i_α. □

Puisque t_{n+1,i_α} et t_{n+2,i_α} ($\alpha \in \{1, ..., n\}$) sont les seules demandes de trafic qui peuvent être optiquement routées (voir lemme 2.1) et puisque, pour chaque $\alpha \in \{1, ..., n\}$ il est possible de router optiquement la demande de trafic du sommet $n+1$ vers le sommet i_α, ou du sommet $n+2$ vers le sommet i_α, (voir lemme 2.3), mais il n'est pas possible de router ces deux demandes de trafic optiquement dans la même solution (voir lemme 2.2), nous avons seulement besoin de considérer les solutions dans lesquelles il y a une demande de trafic optiquement routée entre exactement un des deux sommets $n+1$ et $n+2$, et chacun des sommets i_α, afin de résoudre le problème de décision du groupage de trafic sur notre instance.

Soit X une solution candidate au problème du sac-à-dos. X est un vecteur $\{x_1, \ldots, x_n\}$ dans lequel $x_i = 1$ si l'élément i est choisit ($x_i = 0$ sinon). Considérons la solution du problème de groupage de trafic dans laquelle X (respectivement $\overline{X}$) est le vecteur qui indique les demandes de trafic routées optiquement à partir du sommet $n+1$ (respectivement $n+2$). Ainsi, pour tout $\alpha \in \{1, \ldots, n\}$, t_{n+1,i_α} (respectivement t_{n+2,i_α}) est routée optiquement si et seulement si $x_\alpha = 1$ (respectivement $x_\alpha = 0$). Pour des soucis de lisibilité dans les inégalités suivantes, les sommets i_α sont numérotés de 1 à n : soit $\alpha \in \{1, ..., n\}$, nous avons $i_\alpha = \alpha$. En partant de la condition de satisfiabilité du problème de sac-à-dos, nous obtenons :

$$\begin{aligned}
&\textstyle\sum_{i=1}^n x_i w_i \leq B \\
\Leftrightarrow &\textstyle\sum_{i=1}^n x_i(C - t_{n+1,i}) \leq \sum_{i=1}^n (C - t_{n+1,i}) - (t_{n+1,n+4} + \sum_\beta t_{n+1,k_\beta}) \\
\Leftrightarrow &\textstyle\sum_{i=1}^n (\overline{x_i} t_{n+1,i}) + (t_{n+1,n+4} + \sum_\beta t_{n+1,k_\beta}) \leq (n - \sum_{i=1}^n x_i)C
\end{aligned}$$

Cette dernière inégalité signifie que la quantité de trafic routé électroniquement (le terme gauche de l'inégalité) doit être inférieure ou égale à la capacité d'un lien, C, multipliée par le nombre de liens disponibles (i.e. n moins le nombre de demandes de trafic qui sont routées optiquement).

$$\begin{aligned}
&\textstyle\sum_{i=1}^n x_i v_i \geq K \\
\Leftrightarrow &\textstyle\sum_{i=1}^n x_i(t_{n+1,i} - t_{n+2,i}) \geq Q - \sum_{i=1}^n t_{n+2,i} \\
\Leftrightarrow &\textstyle\sum_{i=1}^n (x_i t_{n+1,i} + \overline{x_i} t_{n+2,i}) \geq Q
\end{aligned}$$

Cette inégalité signifie que la quantité totale de trafic routé optiquement doit être supérieure ou égale à Q.

Ainsi, un vecteur X satisfait le problème de groupage de trafic pour notre instance si et seulement si ce vecteur satisfait le problème du sac-à-dos. Ainsi le problème de groupage de trafic est satisfiable si et seulement si le problème de sac-à-dos est statisfiable.

Théorème 2.1 *Les versions de décision des versions de minimisation et de maximiation du problème de groupage de trafic dans une étoile passive, sont NP-complètes.*

Preuve : Nous avons déjà montré que la version de décision de la version de minimisation du problème de groupage de trafic dans une étoile passive est un problème NP-complet. Puisque nous pouvons facilement passer d'une version à une autre : $OPT_{max} = (\sum_{i \in V_1, j \in V_2} t_{ij}) - OPT_{min}$ (où OPT_{max} est la solution optimale de la version de maximisation et OPT_{min} est la solution optimale de la version de minimisation), la version de décision de la version de maximisation du problème de groupage dans une étoile passive est également NP-complète. □

2.3 Un algorithme polynomial dans le cas où il y a deux longueurs d'ondes par fibre

Nous montrons maintenant que le problème de groupage de trafic dans une étoile passive est un problème que l'on peut résoudre en un temps polynomial si le nombre de longueurs d'ondes dans chaque fibre, W, est égal à 2. Nous donnons pour cela un algorithme qui résout ce problème en un temps polynomial.

Nous rappellons que T est la matrice des demandes de trafic. Tout d'abord, nous remarquons que dans chaque ligne (ou colonne) de T dans laquelle il y a au moins trois valeurs non nulles, il est possible de router optiquement au plus une demande de trafic. En effet, chaque demande de trafic routée optiquement nécessite une longueur d'onde à elle seule, et il n'y a que deux longueurs d'ondes disponibles par fibre. Par contre, quand une ligne (ou colonne) contient seulement deux valeurs différentes de 0, il est éventuellement possible de router ces deux requêtes optiquement. Nous transformons la matrice T en une matrice T' dans laquelle il est possible de router optiquement au plus une demande de trafic de chaque ligne, et une demande de trafic de chaque colonne : si une ligne i de T contient exactement deux valeurs t_{ij} et $t_{ij'}$ non nulles, nous créons dans T' deux lignes $i1$ et $i2$ telles que $t'_{i1,j} = t_{ij}$, $t'_{i2,j} = t_{ij'}$ et les autres valeurs dans ces lignes sont égales à 0. De même, si une colonne j de T contient exactement deux valeurs t_{ij} et $t_{i'j}$ non nulles, nous créons dans T' deux colonnes $j1$ et $j2$ telles que $t'_{i,j1} = t_{ij}$, $t'_{i,j2} = t_{i'j}$ et les autres valeurs dans ces lignes sont égales à 0. De cette manière, il y a dans T' au plus une demande de trafic par ligne et une demande de trafic par colonne qui puisse être optiquement routée.

Nous transformons maintenant notre matrice T' en une autre matrice $M = [m_{ij}]$, dans laquelle nous chercherons ensuite un couplage de poids maximum. Pour cela, nous appliquons la règle suivante : si une demande de trafic t'_{ij} ne peut pas être routée optiquement (i.e. $\sum_{k \neq j} t'_{ik} > C$ ou $\sum_{k \neq i} t'_{kj} > C$), alors $m_{ij} = -\infty$. Dans le cas contraire, $m_{ij} = t'_{ij}$. Puisqu'il y a dans M au plus une demande de trafic par ligne et une demande de trafic par colonne qui peuvent être optiquement routées, et puisque la demande de trafic m_{ij} est différente de $-\infty$ si et seulement si elle peut éventuellement être routée optiquement, le résultat d'un couplage de poids maximum dans le graphe biparti dont M est la matrice d'adjacence, est une solution optimale pour le problème de groupage de trafic dont la matrice de trafic est T.

Théorème 2.2 *Les versions de minimisation et de maximisation du problème de groupage de trafic dans une étoile passive peuvent être résolues en un temps polynomial si le nombre de longueurs d'ondes par fibre est égal à deux.*

Preuve : Une solution optimale de la version de minimisation est aussi une solution optimale du problème de maximisation (et vice-versa), et l'algorithme ci-dessus résout en un temps polynomial ces problèmes. □

Exemple. Considérons l'instance suivante. Le nombre d'émetteurs et le nombre de récepteurs est égal à 3, la capacité de chaque longueur d'onde est 5, et le nombre de longueurs d'onde par fibre est égal à 2. Soit T la matrice de trafic :

$$\mathbf{T} = \begin{pmatrix} 4 & 2 & 1 \\ 3 & 3 & 4 \\ 2 & 0 & 5 \end{pmatrix}$$

Les matrices T' et M créées par notre algorithme sont les suivantes :

$$\mathbf{T'} = \begin{pmatrix} 4 & 2 & 0 & 1 \\ 3 & 0 & 3 & 4 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 \end{pmatrix} \qquad \mathbf{M} = \begin{pmatrix} 4 & 2 & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty \\ -\infty & 0 & 0 & -\infty \\ -\infty & 0 & 0 & 5 \end{pmatrix}$$

Le couplage de poids maximum dans M est $m_{1,1}$ et $m_{4,4}$, qui correspondent à $t_{1,1}$ et $t_{3,3}$. Les demandes de trafic qui sont donc routées optiquement dans une solution optimale sont $t_{1,1}$ et $t_{3,3}$. Les autres demandes de trafic sont routées électroniquement.

2.4 Algorithmes d'approximation

Puisque le problème de groupage de trafic dans une étoile passive est NP-difficile dans le cas général, comme nous l'avons montré dans la section 2.2, nous nous intéressons maintenant à des algorithmes d'approximation avec garantie de performance pour ce problème. Nous montrons dans la section 2.4.1 qu'il n'est pas possible de déduire d'un algorithme d'approximation pour la version de minimisation du problème un algorithme d'approximation pour la version de maximisation du problème (et vice-versa). Dans la section 2.4.2 nous donnons un algorithme d'approximation pour la version de minimisation du problème, et dans la section 2.4.3 nous donnons un algorithme d'approximation pour la version de maximisation du problème.

2.4.1 L'approximation n'est pas conservée entre les versions de minimisation et de maximisation

Théorème 2.3 *Il n'est pas possible de déduire un algorithme d'approximation avec garantie de performance pour la version de maximisation (respectivement minimisation) du problème de groupage de trafic dans une étoile passive, à partir d'un algorithme d'approximation avec garantie de performance pour la version de minimisation (respectivement maximisation).*

Preuve : Soit OPT_{max} le coût d'une solution optimale de la version de maximisation (i.e. la quantité maximum de trafic qui peut être routé optiquement dans une solution réalisable) et OPT_{min} le coût d'une solution optimale de la version de minimisation (i.e. la quantité minimum de trafic routée électroniquement dans une solution réalisable). Soit S une solution du problème de groupage de trafic dans une étoile passive. Nous notons $c_{max}(S)$ le coût de S pour la version de maximisation (i.e. $c_{max}(S)$ est la quantité de trafic qui est routé optiquement dans la solution S) et $c_{min}(S)$ le coût de S pour la version de minimisation (i.e. $c_{min}(S)$ est la quantité de trafic qui est routé électroniquement dans la solution S).

Soient ε_1 et ε_2 deux nombres réels tels que $0 \leq \varepsilon_1 < 1$ et $0 \leq \varepsilon_2$. Montrons qu'il n'est pas possible de déduire un algorithme $(1-\varepsilon_1)$-approché pour la version de maximisation à partir d'un algorithme $(1+\varepsilon_2)$-approché pour la version de minimisation. Soit I une instance du problème dans laquelle on peut router au plus une demande de trafic (égale à 1) optiquement ($OPT_{max} = 1$ et $OPT_{min} = (\sum_{i \in V_1, j \in V_2} t_{ij}) - 1$). Considérons une solution S du problème de groupage de trafic sur l'instance I, dans laquelle toutes les demandes de trafic sont routées électroniquement ($c_{max}(S) = 0$ et $c_{min}(S) = \sum_{i \in V_1, j \in V_2} t_{ij}$). Pour une valeur assez grande de $\sum_{i \in V_1, j \in V_2} t_{ij}$, nous avons $\frac{c_{min}(S)}{OPT_{min}} = \frac{\sum_{i \in V_1, j \in V_2} t_{ij}}{(\sum_{i \in V_1, j \in V_2} t_{ij})-1} \leq 1+\varepsilon_2$ mais il n'existe pas de $\varepsilon_1 < 1$ tel que $\frac{c_{max}(S)}{OPT_{max}} = \frac{0}{1}$ soit supérieur ou égal à $1-\varepsilon_1$.

Montrons maintenant qu'il est possible d'avoir une instance telle que $\sum_{i \in V_1, j \in V_2} t_{ij}$ est aussi grande que l'on le souhaite et dans laquelle on peut router au plus une demande de trafic optiquement : considérons l'instance dans laquelle nous avons $2W$ émetteurs, 2 récepteurs, et la matrice de trafic $[t_{ij}]$ est telle que $t_{1,1} = 1$; $\forall i \in \{2, 3, ..., 2W\},\ t_{i,1} = 0$; et $\forall i \in \{1, ..., 2W\},\ t_{i,2} = \frac{C}{2}$. La seule demande de trafic qui peut être routée optiquement est $t_{1,1}$ car le second récepteur reçoit une quantité de trafic égale à WC, la totalité de la bande passante disponible, et chaque demande de trafic est strictement inférieure à C.

De la même manière, montrons qu'il n'est pas possible de déduire un algorithme $(1+\varepsilon_2)$-approché pour la version de minimisation à partir d'un algorithme $(1-\varepsilon_1)$-approché pour la version de maximisation. Considérons une instance I du problème dans laquelle toutes les demandes de trafic peuvent être routées optiquement ($OPT_{max} = \sum_{i \in V_1, j \in V_2} t_{ij}$ et $OPT_{min} = 0$). Il est trivial qu'une telle instance existe. Considérons une solution S de I dans laquelle toutes les demandes de trafic, sauf deux (de longueurs 1), sont routées optiquement ($c_{max}(S) = (\sum_{i \in V_1, j \in V_2} t_{ij}) - 2$ et $c_{min}(S) = 2$). Pour une valeur assez grande de $\sum_{i \in V_1, j \in V_2} t_{ij}$, nous avons $\frac{c_{max}(S)}{OPT_{max}} = \frac{(\sum_{i \in V_1, j \in V_2} t_{ij})-2}{\sum_{i \in V_1, j \in V_2} t_{ij}}$ $\geq 1-\varepsilon_1$ mais il n'existe pas de $\varepsilon_2 \geq 0$ tel que $\frac{c_{min}(S)}{OPT_{min}} = \frac{2}{0}$ soit inférieur ou égal à $1+\varepsilon_2$. $\square$

2.4.2 Un algorithme d'approximation pour la version de minimisation

Nous donnons une formulation en programmation linéaire en nombres entiers de la version de minimisation du problème de groupage de trafic dans une étoile passive. Pour cela nous notons $x_{ij} \in \{0, 1\}$ la variable qui indique si la demande de trafic t_{ij} est routée optiquement ($x_{ij} = 0$) ou électroniquement ($x_{ij} = 1$). L'objectif est le suivant :

$$\text{Minimiser} \sum_{i \in V_1, j \in V_2} x_{ij} t_{ij} \tag{2.1}$$

Nous avons deux types de contraintes :
les contraintes sur le nombre de longueurs d'ondes :

$$\forall i \in V_1, \sum_{j \in V_2} x_{ij} \geq |V_2| - W \tag{2.2}$$

$$\forall j \in V_2, \sum_{i \in V_1} x_{ij} \geq |V_1| - W \tag{2.3}$$

les contraintes sur le trafic :

$$\forall i \in V_1, \sum_{j \in V_2} (1 - x_{ij})(C - t_{ij}) \leq WC - \sum_{j \in V_2} t_{ij} \tag{2.4}$$

$$\forall j \in V_2, \sum_{i \in V_1} (1 - x_{ij})(C - t_{ij}) \leq WC - \sum_{i \in V_1} t_{ij} \tag{2.5}$$

Les inégalités (2.2) (respectivement (2.3)) signifient qu'au plus W demandes de trafic par émetteur (respectivement récepteur) peuvent être routées optiquement, car chaque demande de trafic routée optiquement occupe une longueur d'onde à elle seule. Les inégalités (2.4) et (2.5) signifient que la bande passante non utilisée $(C - t_{ij})$ quand t_{ij} est routée optiquement, doit être inférieure ou égale à la bande passante disponible (i.e. la quantité totale de bande passante moins la quantité totale des demandes de trafic). Il est équivalent de dire que la quantité de trafic routé électroniquement doit être inférieur à la capacité d'un lien, C, multipliée par le nombre de liens disponibles (i.e. W moins le nombre de demandes de trafic qui sont routées optiquement).
Les contraintes (2.4) et (2.5) sont équivalentes à :

$$\forall i \in V_1, \sum_{j \in V_2} x_{ij}(C - t_{ij}) \geq C(|V_2| - W) \tag{2.6}$$

$$\forall j \in V_2, \sum_{i \in V_1} x_{ij}(C - t_{ij}) \geq C(|V_1| - W) \tag{2.7}$$

Les contraintes sur le trafic impliquent les contraintes sur le nombre de longueurs d'ondes : si nous divisons par C les contraintes sur le nombre de longueurs d'ondes, nous obtenons :

$$\forall i \in V_1, \sum_{j \in V_2} x_{ij}(1 - \frac{t_{ij}}{C}) \geq |V_2| - W \tag{2.8}$$

$$\forall j \in V_2, \sum_{i \in V_1} x_{ij}(1 - \frac{t_{ij}}{C}) \geq |V_1| - W \tag{2.9}$$

Puisque $t_{ij} \leq C$, ces dernières inégalités impliquent les contraintes sur le nombre de longueurs d'ondes (2.2) et (2.3).

Nous pouvons donc formuler notre problème de la façon suivante :

$$\text{Minimiser} \sum_{i \in V_1, j \in V_2} x_{ij} t_{ij} \tag{2.10}$$

sous les contraintes :

$$\forall i \in V_1, \sum_{j \in V_2} x_{ij}(C - t_{ij}) \geq C(|V_2| - W) \tag{2.11}$$

$$\forall j \in V_2, \sum_{i \in V_1} x_{ij}(C - t_{ij}) \geq C(|V_1| - W) \tag{2.12}$$

$$\forall i \in V_1, j \in V_2, x_{ij} \in \{0, 1\} \tag{2.13}$$

Théorème 2.4 *Soit H_i le $i^{\text{ème}}$ nombre harmonique : $H_i = 1 + \frac{1}{2} + ... + \frac{1}{i}$. Soit C la capacité d'une longueur d'onde. Il existe un algorithme qui s'exécute en un temps polynomial et qui est $H_{2\,C}$-approché pour la version de minimisation du problème de groupage de trafic dans une étoile passive.*

Preuve : Nous transformons la version de minimisation du problème de groupage de trafic dans une étoile passive en un problème de multi-couverture contrainte de multi-ensemble. Nous donnons pour cela quelques définitions. Étant donné un ensemble d'éléments $\mathcal{U}$, une collection T de sous-ensembles de $\mathcal{U}$: $T = \{S_1, S_2, ..., S_k\}$, et une fonction de coût $c : T \to Q^+$, le problème de *couverture d'ensemble* consiste à trouver un ensemble de sous-ensembles $\mathcal{C} \subseteq T$ de coût minimum qui couvre tous les éléments de $\mathcal{U}$ (i.e. $\bigcup_{e \in \mathcal{C}} e = \mathcal{U}$). Le problème de *multi-couverture de multi-ensemble* est une généralisation naturelle du problème de couverture d'ensemble : dans ce problème, chaque élément $e \in \mathcal{U}$ apparaît dans un multi-ensemble S un nombre quelconque de fois, noté $m(S, e)$, et à chaque élément e est associé un nombre entier r_e, qui indique le nombre minimum de fois que l'élément e doit être couvert. Dans le problème de *multi-couverture contrainte de multi-ensemble*, chaque sous-ensemble $S \in T$ est choisi au plus une fois. Nous pouvons donc formuler ce problème sous forme du programme linéaire en nombres entiers suivant :

$$\text{Minimiser} \sum_{S \in T} c(S) x_S, \text{ sous les contraintes } : \forall e \in \mathcal{U} \sum_{S \in T} m(S, e) x_S \geq r_e \text{ et } x_S \in \{0, 1\}.$$

Montrons maintenant la réduction du problème de groupage de trafic en un problème de multi-couverture contrainte de multi-ensemble. Cette transformation vient directement de la formulation de notre problème en un programme linéaire en nombres entiers : pour chaque demande de trafic de l'émetteur $i \in V_1$, noté e_i, vers le récepteur $j \in V_2$, noté r_j, nous créons le sous-ensemble S_{ij} qui contient $(C - t_{ij})$ fois l'élément e_i et $(C - t_{ij})$ fois l'élément r_j. Le coût de ce sous-ensemble est $c(S_{ij}) = t_{ij}$. Chaque élément $e \in V_1$ doit être couvert $r_e = C\,(|V_2| - W)$ fois, et chaque élément $e \in V_2$ doit être couvert $r_e = C\,(|V_1| - W)$ fois.

S. Rajagopalan et V. Vazirani donnent dans [70] un algorithme d'approximation glouton pour le problème de multi-couverture contrainte de multi-ensemble. Cet algorithme consiste à prendre à chaque étape l'ensemble de T ayant le meilleur rapport coût-efficacité, et à retirer cet ensemble de T. Le rapport coût-efficacité d'un ensemble S est égal au coût moyen auquel il couvre de nouveaux éléments, i.e. c'est le coût de S divisé par le nombre de ses éléments qui ne sont pas encore couverts. Cet algorithme a un rapport d'approximation égal à H_k [70], où H_k est le $k^{\text{ème}}$ nombre harmonique (i.e. $H_k = 1 + \frac{1}{2} + ... + \frac{1}{k}$), et où k est la taille du plus grand multi-ensemble de l'instance donnée. Dans notre cas, la taille d'un multi-ensemble S_{ij} est $2\,C - 2t_{ij}$, ce qui est inférieur à $2\,C$. Nous obtenons donc une solution du problème de groupage de trafic qui a un rapport d'approximation de $H_{2\,C} \leq log(2\,C) + 1$. □

Exemple. Considérons le problème dans lequel le nombre d'émetteurs et le nombre de récepteurs sont égaux à 3, la capacité de chaque longueur d'onde est 5, et le nombre de longueurs d'ondes par fibre est 2. Soit T la matrice de trafic :

$$\mathbf{T} = \begin{pmatrix} 4 & 2 & 1 \\ 3 & 3 & 4 \\ 2 & 0 & 5 \end{pmatrix}$$

Nous obtenons les multi-ensembles suivants :
$S_{1,1} = \{e_1, r_1\}$ et $c(S_{1,1}) = 4$
$S_{1,2} = \{e_1, e_1, e_1, r_2, r_2, r_2\}$ et $c(S_{1,2}) = 2$
$S_{1,3} = \{e_1, e_1, e_1, e_1, r_3, r_3, r_3, r_3\}$ et $c(S_{1,3}) = 1$
$S_{2,1} = \{e_2, e_2, r_1, r_1\}$ et $c(S_{2,1}) = 3$
$S_{2,2} = \{e_2, e_2, r_2, r_2\}$ et $c(S_{2,2}) = 3$
$S_{2,3} = \{e_2, r_3\}$ et $c(S_{2,3}) = 4$
$S_{3,1} = \{e_3, e_3, e_3, r_1, r_1, r_1\}$ et $c(S_{3,1}) = 2$
$S_{3,2} = \{e_3, e_3, e_3, e_3, e_3, r_2, r_2, r_2, r_2, r_2\}$ et $c(S_{3,2}) = 0$
$S_{3,3} = \{\emptyset\}$ et $c(S_{3,3}) = 5$

La solution retournée par l'algorithme glouton pour le problème de multi-couverture contrainte de multi-ensemble [70] est la suivante : $\mathcal{C} = \{S_{1,2}, S_{1,3}, S_{2,1}, S_{2,2}, S_{2,3}, S_{3,1}, S_{3,2}\}$. Les demandes de trafic routées électroniquement sont $T = \{t_{1,2}, t_{1,3}, t_{2,1}, t_{2,2}, t_{2,3}, t_{3,1}, t_{3,2}\}$; les autres ($t_{1,1}$ et $t_{3,3}$) seront routées optiquement.

2.4.3 Un algorithme d'approximation pour la version de maximisation

Nous donnons une formulation en programmation linéaire en nombres entiers de la version de maximisation du problème de groupage de trafic dans une étoile passive.

$$\text{Maximiser} \sum_{i \in V_1, j \in V_2} y_{ij} t_{ij} \tag{2.14}$$

sous les contraintes :

$$\forall i \in V_1, \sum_{j \in V_2} y_{ij}(C - t_{ij}) \leq CW - \sum_{j \in V_2} t_{ij} \tag{2.15}$$

$$\forall j \in V_2, \sum_{i \in V_1} y_{ij}(C - t_{ij}) \leq CW - \sum_{i \in V_1} t_{ij} \tag{2.16}$$

$$\forall i \in V_1, j \in V_2, y_{ij} \in \{0, 1\} \tag{2.17}$$

Ici y_{ij} indique si la demande de trafic t_{ij} est routée optiquement ($y_{ij} = 1$) ou électroniquement ($y_{ij} = 0$). Le programme linéaire en nombres entiers de ce problème est obtenu en remplaçant x_{ij} dans le programme linéaire en nombres entiers de la version de minimisation par $1 - y_{ij}$.

Théorème 2.5 *Il existe un algorithme randomisé qui s'exécute en un temps polynomial et qui est* $(2 + \frac{4}{5})$*-approché pour la version de maximisation du problème de groupage de trafic dans une étoile passive.*

Preuve :Nous transformons ce problème en un problème de couplage avec demandes ("demand matching problem" en anglais) [74]. Le problème de couplage avec demandes est le suivant : soit un graphe $G = (V, E)$ dans lequel chaque sommet $v \in V$ a une *capacité*, qui est un nombre entier et est notée par b_v. Chaque arête $e = (u, v) \in E$ a une *demande*, qui est un nombre entier et est notée par d_e. De plus, à chaque arête $e \in E$ est associée un *profit*, noté par p_e. Un *couplage avec demandes* est un sous-ensemble $M \subseteq E$ tel que $\sum_{e \in \delta(v) \cap M} d_e \leq b_v$ pour chaque sommet v. Ici $\delta(v)$ représente l'ensemble des arêtes de G qui sont incidentes au sommet v. Le problème de couplage avec demandes consiste à trouver un couplage avec demandes qui maximise $\sum_{e \in M} p_e$. Le programme linéaire en nombres entiers de ce problème est donc :

$$\text{Maximiser} \sum_{e \in E} \frac{p_e}{d_e} x_e, \quad \text{sous les contraintes } : \forall v \in V, \sum_{e \in \delta(v)} x_e \leq b_v \text{ et } \forall e \in E, x_e \in \{0, d_e\}.$$

B. Shepherd et A. Vetta donnent dans [74] un algorithme randomisé $(2 + \frac{4}{5})$-approché pour le problème de couplage avec demandes dans des graphes bipartis.

La transformation de notre problème de groupage de trafic en un problème de couplage avec demandes est la suivante : $\forall e \in (V_1 \times V_2),\ x_e = y_e(C - t_e)\ ,\ p_e = t_e\ ,\ d_e = C - t_e$. $\forall v \in V_1,\ b_v = CW - \sum_{j \in V_2} t_{vj}$ et $\forall v \in V_2,\ b_v = CW - \sum_{i \in V_1} t_{iv}$.

Puisqu'il existe un algorithme $(2 + \frac{4}{5})$-approché pour le problème de couplage avec demandes, il existe également un algorithme $(2 + \frac{4}{5})$-approché pour la version de maximisation du problème de groupage de trafic dans une étoile passive. □

2.5 Conclusion

Nous avons étudié dans ce chapitre le problème de groupage de trafic dans un réseau WDM en étoile, les émetteurs et les récepteurs étant situés aux feuilles de l'étoile. Nous avons montré que ce problème est NP-difficile dans le cas général, et nous avons donné un algorithme polynomial dans le cas où chaque lien possède deux longueurs d'ondes. Nous avons montré que nous ne pouvons pas déduire d'algorithme d'approximation avec garantie de performance pour le problème dans lequel on souhaite minimiser la quantité de trafic routé électroniquement (version de minimisation), à partir d'un algorithme d'approximation avec garantie de performance pour le problème dans lequel on souhaite maximiser la quantité de trafic routé optiquement (version de maximisation). Nous avons ensuite donné un algorithme avec un rapport d'approximation égal à H_{2C} pour la version de minimisation, et un algorithme randomisé avec un rapport d'approximation égal à $(2 + \frac{4}{5})$ pour la version de maximisation.
Ces deux derniers algorithmes sont obtenus grâce à des réductions de notre problème vers d'autres problèmes plus généraux. Il serait donc intéressant de voir si l'on peut tenir compte des spécificités des instances de ces problèmes obtenues après réduction de notre problème, pour obtenir des rapports d'approximation plus petits pour les deux versions de notre problème. De plus, puisque les

solutions retournées par ces algorithmes sont à la fois des solutions du problème de minimisation et du problème de maximisation, il serait intéressant de comparer expérimentalement les résultats obtenus avec ces deux algorithmes.

Enfin, le problème consistant non pas à minimiser la quantité totale de trafic routé électroniquement, mais à minimiser le nombre de demandes de trafic routé électroniquement, pourrait également être une perspective.

Chapitre 3

Un voisinage exponentiel pour le problème de tournées de véhicules

Nous nous intéressons à un Problème de Tournées de Véhicules (PTV). Ce problème est connu pour être NP-difficile [40], et la recherche locale peut alors être utile pour obtenir de bonnes solutions de ce problème. Nous introduisons un *voisinage de taille exponentielle* pour ce problème et nous montrons que ce voisinage peut être exploré en un temps polynomial. Ces travaux ont été présentés dans [14].

3.1 Introduction

Afin de trouver de bonnes solutions à un problème d'optimisation combinatoire NP-difficile, on utilise des heuristiques, qui sont des algorithmes qui - bien que ne proposant pas de garantie de performances - retournent en général de bonnes solutions du problème et un un temps raisonnable (polynomial). Une classe importante d'heuristiques est celle des *algorithmes de recherche locale* [1] (voir Section 1.1.3 pour un rappel sur les algorithmes de recherche locale).

Un algorithme de recherche locale pour un problème d'optimisation combinatoire commence avec une solution réalisable de ce problème, et essaie d'améliorer cette solution en cherchant une bonne solution dans le voisinage de cette solution (puis recommence ensuite avec la solution trouvée). La qualité de l'optimum local trouvé, ainsi que le temps nécessaire pour le trouver, dépendent fortement du voisinage étudié. Un paramètre important du voisinage est sa taille : intuitivement, plus un voisinage est grand (i.e. plus une solution a de solutions voisines), plus la qualité de la meilleure solution du voisinage risque d'être bonne. Cependant, le temps de recherche de la meilleure solution doit rester polynomial afin que l'algorithme de recherche locale soit polynomial. Ainsi un voisinage de taille exponentielle peut être très intéressant s'il est explorable en un temps polynomial. C'est pourquoi de nombreux travaux s'intéressent à des voisinages de tailles exponentielles et essaient d'identifier ceux qui peuvent être explorés en un temps polynomial [2].

De tels travaux ont été effectués pour différents problèmes d'optimisation combinatoire : des problèmes d'ordonnancement [32, 50], le problème du voyageur de commerce [44, 45] et des généralisations telles que le problème de l'affectation quadratique [35]. Il existe pour certains de ces problèmes des voisinages de taille exponentielle qui peuvent être explorés en un temps polynomial (problèmes d'ordonnancement, du voyageur de commerce), alors que pour d'autres comme le problème de l'affectation quadratique, l'exploration de plusieurs voisinages de taille exponentielle mène à des problèmes NP-difficile [35].

Dans ce chapitre, nous nous intéressons au problème de tournées de véhicules (PTV) [78]. Plusieurs voisinages exponentiels explorables en un temps polynomial ont déjà été proposés pour ce problème. Ergun et al. [37] ont proposé plusieurs voisinages exponentiels, et ont montré comment trouver la meilleure solution de ces voisinages grâce à un algorithme qui repose sur la recherche de plus courts chemins contraints dans un graphe auxiliaire (une approche similaire a été utilisée dans [77]). Xu et Kelly [81] ont proposé un algorithme de recherche tabou dans un voisinage de taille exponentiel. La solution sélectionnée dans le voisinage est trouvée grâce à un flot de coût minimum, mais cette solution peut être uniquement une approximation de la meilleure solution du voisinage.

Dans ce chapitre, nous considérons un nouveau voisinage de taille exponentielle qui repose sur un problème de couplage, et qui généralise celui étudié par Gutin pour le problème du voyageur de commerce. Gutin [44] a étudié, pour le problème du voyageur de commerce, un voisinage exponentiel explorable en un temps polynomial en se basant sur un problème de couplage complet dans un graphe biparti. Nous montrons dans la suite de ce chapitre que ce voisinage étudié pour le voyageur de commerce peut également être exploré en un temps polynomial pour le problème de tournées de véhicules.

Plan du chapitre : Dans la partie restante de cette section, nous présentons le problème de tournées de véhicules (PTV) que nous considérons dans ce chapitre (voir section 3.1.1), et un problème de couplage contraint (CCCP$_{min}$) que nous utiliserons par la suite (voir section 3.1.2). Dans la section 3.2, nous définissons le voisinage exponentiel que nous étudions. Dans la Section 3.3 nous étudions le cas où les clients demandent tous la même quantité de biens et nous réduisons le problème qui consiste à trouver la meilleure solution du voisinage, en une instance du problème CCCP$_{min}$. Dans la section 3.4 nous résolvons le problème CCCP$_{min}$. Dans la section 3.5 nous montrons que si les clients demandent chacun une quantité arbitraire de biens, alors la recherche de la meilleure solution du voisinage devient un problème NP-difficile. Nous concluons ce chapitre par la section 3.6.

3.1.1 Le problème de Tournées de Véhicules (PTV)

Le *Problème de Tournées de Véhicules* (PTV) est le suivant : n clients doivent être servis à partir d'un dépôt unique. Chaque client demande une certaine quantité de biens, et un véhicule de capacité C est disponible pour livrer les biens. Puisque la capacité du véhicule est limitée, le véhicule doit périodiquement retourner au dépôt pour se recharger. Il n'est pas possible de livrer les biens d'un même client en plusieurs fois. Une solution du PTV est donc une collection de tours, dans laquelle chaque client est livré une fois et une seule, et la quantité de biens livrée lors d'un même tour ne doit pas excéder la capacité C du véhicule.

Dans la majeure partie de ce chapitre, nous considérerons une version restreinte du PTV dans laquelle la demande d'un client est égale à un (i.e. tous les clients demandent la même quantité de biens). Ainsi, le nombre de clients livrés lors d'un même tour est d'au plus C. Nous appellerons cette variante du PTV, le PTV *avec demandes unitaires*. Dans la dernière partie de ce chapitre, nous considérerons le problème général défini ci-dessus.

Du point de vue de la théorie des graphes, le PTV avec demandes unitaires peut être défini de la façon suivante : soit $G = (V, E)$ un graphe complet orienté avec un ensemble de sommets $V = \{0, 1, 2, ..., n\}$ (qui représentent les clients) et un ensemble d'arcs E. Le sommet 0 représente le dépôt, et chacun des autres sommets représente un client à servir. À chaque arc (i, j) est associée une valeur $d_{i,j}$ représentant la distance (ou temps de parcours) entre i et j. Le but est de trouver un ensemble de tours de longueur (temps de parcours) totale minimum, i.e. on veut minimiser la distance parcourue (ou le temps de parcours) du véhicule. Chaque tour commence et se termine au sommet 0 (le dépôt), chaque sommet doit être visité exactement une fois, et le nombre de sommets par tour doit être inférieur ou égal à C. Par la suite nous considérons que l'ensemble des distances entre les sommets est stocké dans une *matrice des distances*. Notons que nous ne faisons aucune hypothèse sur ces distances. Celles ci peuvent ou non être symétriques ($\forall \{i, j\} \in V, d_{i,j} = d_{j,i}$), et peuvent ou non respecter l'inégalité triangulaire ($\forall \{i, j, k\} \in V, d_{i,k} \leq d_{i,j} + d_{j,k}$).

	0	1	2	3	4	5	6	7
0	0	2	1	4	4	1	4	2
1	4	0	3	4	1	4	4	4
2	4	4	0	2	4	4	1	4
3	1	4	4	0	1	4	4	4
4	1	4	4	2	0	4	4	4
5	4	1	4	4	4	0	3	4
6	4	4	4	4	4	4	0	1
7	2	4	4	4	4	4	4	0

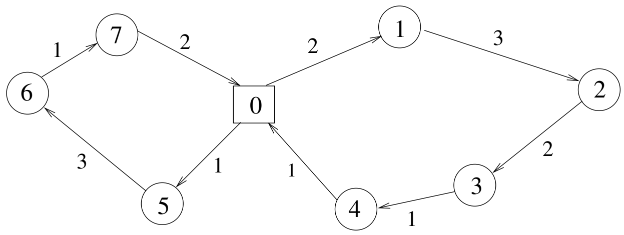


FIG. 3.1 – *Gauche* : Matrice des distances. *Droite* : Une solution du PTV avec des demandes unitaires.

Exemple 1 : La figure 3.1 montre une instance du PTV avec demandes unitaires dans laquelle il y a 7 clients. La capacité du véhicule est 4 et la matrice des distances est montrée sur la figure 3.1 *Gauche*. Une solution de coût 16 est montrée sur la figure 3.1 *Droite* (le coût est la longueur totale des tours, i.e. la somme des longueurs des arcs du tour). Il y a 2 tours : dans le premier tour, le véhicule livre les clients 1, 2, 3, et 4 puis retourne au dépôt, et dans le deuxième tour le véhicule livre les clients 5, 6, et 7 puis retourne au dépôt.

3.1.2 Le problème du Couplage Complet Contraint par Partitions (CCCP)

Le problème de *Couplage Complet Contraint* (CCC) est défini de la façon suivante : Dans un graphe *biparti* $G(V, E)$, i.e. un graphe dans lequel l'ensemble des sommets V peut être partitionné en deux ensembles disjoints, V_1 et V_2 tels que chaque arête de E relie un sommet de V_1 à un sommet de V_2, un ensemble $M \subseteq E$ est un *couplage* si aucun sommet de V n'est incident à plus d'une arête de M. La taille d'un couplage M est le nombre d'arêtes qu'il contient. Si $|V_1| \leq |V_2|$ et chaque sommet $x \in V_1$ est incident à une arête de M, alors le couplage est dit *complet*. Soient $E_1, E_2, \ldots, E_k$ des sous-ensembles de E, et $r_1, r_2, \ldots, r_k$ des entiers positifs. Le problème du *couplage complet contraint* (CCC) consiste à déterminer s'il existe un couplage complet M de G qui satisfasse les contraintes suivantes :

$$|M \cap E_j| \leq r_j \ \ \forall j \in \{1, \ldots, k\}.$$

Il a été montré dans [52] que ce problème est NP-complet (grâce à une réduction du problème de satisfaisabilité des expressions booléennes). Dans le cas où il y a une seule contrainte, les auteurs donnent un algorithme qui le résout en un temps polynomial.

Nous sommes intéressés par le problème CCC dans lequel l'ensemble des sous-ensembles $E_1, E_2, \ldots, E_k$ n'est plus arbitraire mais correspond à une partition de V_2. Soit $T_1, T_2, \ldots, T_k$ une partition de V_2. Pour chaque $j \in \{1, \ldots, k\}$, soit E_j l'ensemble des arêtes de E qui contiennent un sommet de T_j. Le problème de *Couplage Complet Contraint par Partitions* (CCCP) consiste à déterminer s'il existe un couplage complet M de G qui satisfasse les contraintes suivantes :

$$|M \cap E_j| \leq r_j \ \ \forall j \in \{1, \ldots, k\}.$$

Dans la version pondérée du problème CCCP, chaque arête $[v, w]$ a un poids c_{vw}. Notre but est de trouver un CCCP de poids minimum. Ce problème sera appelé le problème du *couplage complet contraint par partitions et de poids minimum* par la suite, et sera noté CCCP$_{min}$. Comme souvent, nous noterons m le nombre d'arêtes de G.

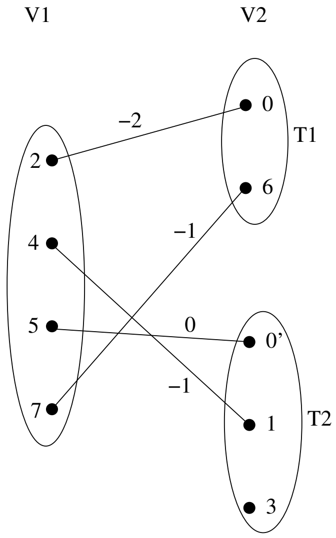
Exemple 2 : La figure 3.2 montre un exemple d'une instance du problème CCCP de poids minimum : nous avons un graphe (complet) biparti avec $V_1 = \{2, 4, 5, 7\}$, $V_2 = T_1 \cup T_2$, $T_1 = \{0, 6\}$ et $T_2 = \{0', 1, 3\}$. La figure 3.2 *Droite* représente la matrice des distances (ou la matrice des coûts), i.e. elle indique la distance (le coût) entre les sommets de V_1 et les sommets de V_2. Ici, $r_1 = r_2 = 2$. La figure 3.2 *Gauche* représente une solution optimale du problème CCCP$_{min}$: les arêtes $\{[2, 0], [4, 1], [5, 0'], [7, 6]\}$ forment un couplage complet contraint par partitions de coût -4.

3.2 Un voisinage exponentiel pour le PTV

Considérons l'algorithme de recherche locale suivant :

Nous commençons avec une solution réalisable (quelconque) S du PTV. Dans cette solution, nous choisissons un sous-ensemble de clients, qui seront considérés comme *mobiles*. Les autres clients et le dépôt seront *fixes*. Nous appellerons sommets l'ensemble des clients et le dépôt. Une *solution voisine* S' de S est une solution dans laquelle chaque sommet mobile a été inséré entre deux sommets fixes de la solution initiale (ces deux sommets peuvent éventuellement être tout les deux le dépôt dans le cas où le seul sommet fixe d'un tour est le dépôt).

Nous introduisons maintenant quelques définitions utiles par la suite. Dans S, chaque sommet fixe u a un *successeur* $s(u)$ qui est le sommet fixe qui le suit dans le tour (si u est le dépôt et est le seul sommet fixe du tour, alors $s(u) = u$). Par exemple, si S est la solution représentée sur la figure 3.3 *Gauche*, le successeur $s(1)$ du sommet 1, est 3. C'est le successeur direct de 1 quand on a retiré les sommets mobiles (voir figure 3.3 *Droite*). Étant donnés une solution S du PTV (i.e.



	0	6	0'	1	3
2	-2	4	3	1	7
4	4	1	6	-1	1
5	0	4	0	4	7
7	1	-1	3	4	4

FIG. 3.2 – Une instance du problème CCCP de poids minimum, avec $V_1 = \{2, 4, 5, 7\}$, $V_2 = \{0, 0', 1, 3, 6\}$, $T_1 = \{0, 6\}$, $T_2 = \{0', 1, 3\}$ et $r_1 = r_2 = 2$. *Gauche* : Une solution optimale de coût -4. *Droite* : Matrice des distances.

un ensemble de tours), et un un ensemble de sommets mobiles (parmi les sommets de S), nous appellerons *graphe des sommets fixes* de S le graphe correspondant au graphe S dans lequel on retire les sommets mobiles et les arcs qui leur sont associés, et on rajoute s'il n'existe pas déjà un arc de chaque sommet fixe u vers son successeur $s(u)$ (le coût de cet arc est égal à la distance entre u et $s(u)$ telle que donnée par la matrice des distances).

Exemple 3 : La figure 3.3 (*à gauche*) montre une solution initiale de coût 16. Les sommets fixes forment le graphe des sommets fixes (*à droite*), dans lequel on peut insérer au plus un sommet mobile entre deux sommets (fixes). La figure 3.4 montre une solution voisine de la solution montrée sur la figure 3.3 *à gauche*. Le coût de cette nouvelle solution est 11 (la matrice des distances est celle donnée dans la figure 3.1). L'orientation de chaque tour est préservée lors de la construction de la solution voisine.

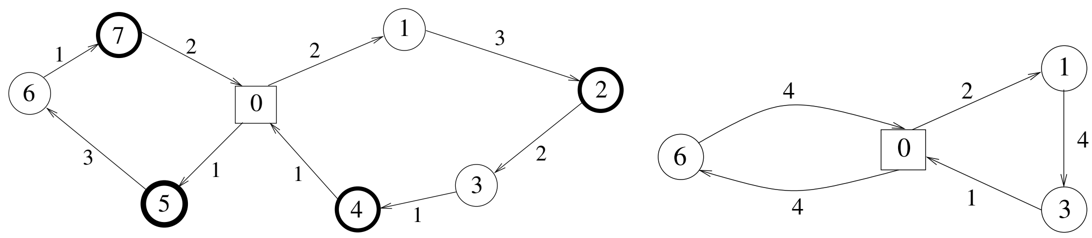


FIG. 3.3 – *Gauche* : une solution du PTV. Les sommets mobiles sont entourés en gras. *Droite* : le graphe correspondant, dans lequel les sommets mobiles ont été retirés.

Remarques : Comme on peut insérer au plus un sommet entre deux sommets fixes, le nombre de sommets mobiles doit être inférieur ou égal au nombre d'arcs dans le graphe des sommets fixes,

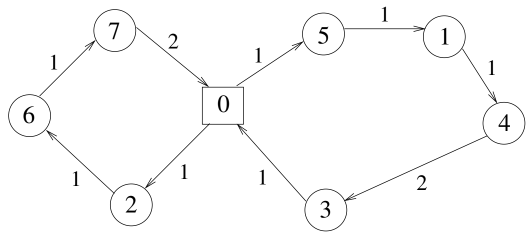


FIG. 3.4 – Une solution de coût 11 pour le PTV.

qui est inférieur à deux fois le nombre de sommets fixes. Nous pouvons également remarquer que le nombre de tours ne peut pas augmenter : il est constant, ou diminue dans le cas où le seul sommet fixe d'un tour est le dépôt et aucun sommet mobile n'a été inséré dans ce tour.

Nous ne nous préoccupons pas ici de la manière dont sont choisis les sommets mobiles. L'ensemble des sommets mobiles peut par exemple être choisi de manière aléatoire parmi l'ensemble des sommets (le dépôt excepté). Il est facile de vérifier si un couple (solution initiale, ensemble des sommets mobiles) a un voisinage non-vide. Ceci est le cas si et seulement si les deux conditions suivantes sont remplies. Tout d'abord, il doit être possible d'insérer chaque sommet mobile entre deux sommets fixes, i.e. le nombre d'arcs dans le graphe des sommets fixes doit être supérieur ou égal au nombre de sommets mobiles. De plus, il faut que dans une solution voisine la somme des demandes des clients de chaque tour (le nombre de clients dans le cas de demandes unitaires) soit inférieure ou égale à la capacité C du véhicule. Dans le cas du PTV avec demandes unitaires, pour chaque tour dans le graphe des sommets fixes, le nombre maximum de sommets mobiles qu'il est possible d'insérer dans ce tour est égal au minimum entre, d'une part le nombre d'arcs de ce tour, et d'autre part la différence entre C et le nombre de sommets fixes (le dépôt excepté) du tour.

Nous appellerons le voisinage défini ci-dessus le *voisinage multi-insertions*. Nous considérons un algorithme de recherche locale utilisant une plus profonde descente : nous cherchons donc à chaque étape la meilleure solution voisine de la solution courante (s'il y en a plusieurs nous en choisissons une au hasard). Cette solution devient alors la solution courante de l'étape suivante. Afin d'avoir un algorithme efficace, il faut que le le temps nécessaire pour trouver la meilleure solution voisine à chaque étape soit polynomial.

Théorème 3.1 *Dans le voisinage multi-insertions, le nombre de solutions voisines d'une solution peut être exponentiel, même pour le* PTV *avec des demandes unitaires.*

Preuve : Considérons l'instance suivante : nous avons n clients à servir (où n est un nombre pair) et la capacité du véhicule est d'au moins n. Chaque client a une demande égale à 1. Supposons que la solution initiale soit une solution dans laquelle il y a seulement un tour, et que nous ayons $\frac{n}{2}$ sommets mobiles. Le nombre de solutions voisines est dans ce cas $(\frac{n}{2}+1)!$ □

Puisque le nombre de solutions voisines peut être exponentiel, nous ne pouvons pas (en temps polynomial) énumérer toutes les solutions possibles et ensuite choisir la meilleure. Afin d'avoir un algorithme qui nous retourne la meilleure solution voisine en temps polynomial, nous allons maintenant réduire la recherche de la meilleure solution dans un voisinage multi-insertions en un problème de couplage contraint par partitions et de poids minimum (nous montrerons ensuite que ce nouveau problème peut être résolu en temps polynomial).

3.3 Recherche de la meilleure solution du voisinage : réduction vers le problème CCCP$_{min}$

Dans cette section, nous nous intéressons uniquement au cas où les demandes des clients sont unitaires. Nous dirons que, étant donnés une solution S et un ensemble de sommets mobiles, S' est une solution voisine de S si et seulement si S' peut être obtenue par une étape de l'algorithme de recherche locale énoncé ci-dessus : S' est obtenue à partir de S en retirant les sommets mobiles, puis en insérant chaque sommet mobile entre deux fixes. S' est la meilleure solution voisine de S si et seulement si S' est une solution voisine de S et il n'y a pas d'autre solution voisine de S qui a un coût (une distance totale des tours) plus petit que le coût de S'.

Soit une instance du problème de la recherche de la meilleure solution dans un voisinage multi-insertions : une matrice des distances D (dans laquelle $d_{i,j}$ est la distance de i à j), un véhicule de capacité C, une solution initiale réalisable du PTV avec demandes unitaires, notée S (et représentée par un graphe $G(V, E)$), et soit une partition de l'ensemble des clients en un ensemble des clients mobiles et un ensemble des clients fixes. Soit k le nombre de tours de S. Soit u un sommet fixe de S, nous rappelons que le successeur fixe de u dans S est noté $s(u)$.

Nous allons maintenant transformer la recherche de la meilleure solution du voisinage de S en un problème de couplage. Soit $G'(V', E')$ un graphe complet biparti tel que $V' = V_1 \cup V_2$, où les sommets de V_1 sont les clients mobiles et les sommets de V_2 sont les clients fixes plus k copies du dépôt. Nous avons alors $E' = \{[u, v] \mid u \in V_1, v \in V_2\}$. Les poids des arêtes sont définis de la façon suivante : soit $u \in V_1$ et $v \in V_2$, le poids de l'arête $[u, v]$ est égal à $d_{v,u} + d_{u,s(v)} - d_{v,s(v)}$. Si une arête $[u, v]$ appartient au couplage obtenu, alors cela signifie que le sommet u est inséré entre les sommets v et $s(v)$. Soit $\{T_1, T_2, \ldots, T_k\}$ une partition de V_2 telle que, pour chaque $i \in \{1, \ldots, k\}$, les sommets de T_i sont les clients fixes du $i^{\text{ème}}$ tour de S et une copie du dépôt. Soit $r_i = C - |T_i| + 1$. Notre but est de trouver dans G' un couplage complet de poids minimum M tel que le nombre d'arêtes dans $M \cap E_i$ est inférieur ou égal à r_i, pour tout $i \in \{1, \ldots, k\}$.

Exemple 4 : La figure 3.2 montre l'instance du problème de couplage correspondant à la solution initiale du PTV montrée dans la figure 3.3. La matrice des distances de G' est montrée dans la figure 3.2 *Droite*. Dans la figure 3.2 *Gauche*, nous n'avons pas dessiné toutes les arêtes mais seulement les quatre arêtes qui appartiennent au couplage complet contraint par partitions et de poids minimum (de coût -4). Cette solution optimale correspond à la solution de la figure 3.4 pour le PTV (la meilleure solution voisine de l'instance représentée sur la figure 3.3) : on commence avec la solution initiale de la figure 3.3, on insère le sommet 2 entre les sommets 0 et 6, et les sommet 7 entre les sommets 6 et 0, dans le premier tour (le tour dont les sommets fixes sont ceux de T_1), et on insère le sommet 4 entre les sommets 1 et 3, et le sommet 5 entre le dépôt (0' est une copie du dépôt) et le sommet 1 dans le deuxième tour.

Théorème 3.2 *Soit S une solution réalisable du* PTV *avec des demandes unitaires, et soit ξ un sous-ensemble des sommets de S définis comme mobiles. Soit G' le graphe biparti qui correspond* *au voisinage multi-insertions de S (G' est construit comme indiqué ci-dessus), et soit M la solution du problème CCCP_{min} sur G'. Soit S' la solution qui correspond à S dans laquelle nous avons retiré les sommets mobiles, puis inséré chaque sommet mobile de la façon suivante : si $[u, v] \in M$, alors le sommet u est inséré entre les sommets v et $s(v)$. S' fait partie des meilleures solutions voisines de S.*

Preuve : Si nous choisissons $[u, v]$ dans M, cela signifie que nous insérons le sommet mobile u entre les sommets fixes v et $s(v)$: le coût de $[u, v]$ est égal à $d_{v,u} + d_{u,s(v)} - d_{v,s(v)}$, ce qui est correspond au coût pour aller de v à u, plus le coût pour aller de u à $s(v)$ ($s(v)$ est le successeur fixe de v), moins le coût pour aller de v à $s(v)$. Ainsi, la somme des coûts des poids des arêtes de M est égale à la somme que l'on doit rajouter à la longueur totale des tours dans le graphe des sommets fixes de G pour obtenir la longueur totale des tours dans S'. Plus cette somme est petite, plus la longueur totale des tours de S' sera petite. Montrons maintenant que toutes les solutions voisines de S peuvent être représentées par un couplage complet contraint par partitions dans G', et que seules les solutions voisines de S sont représentées par un tel couplage.

Nous savons que le nombre de sommets mobiles est inférieur ou égal au nombre d'arcs dans le graphe des sommets fixes (car sinon nous ne pourrions pas insérer chaque sommet mobile entre deux sommets fixes, et S n'aurait pas de solution voisine). Ce nombre est inférieur ou égal au nombre de sommets fixes (sans prendre en compte le dépôt), plus le nombre de tours dans le graphe. En effet, chaque sommet fixe (à part le dépôt), a un et un seul arc sortant, et le nombre d'arcs sortants du dépôt est égal au nombre de tours, k. Le nombre d'arcs dans le graphe des sommets fixes est donc égal à $|V_2|$, et nous en déduisons que $|V_1| \leq |V_2|$. Ainsi, puisque M est un couplage complet de G', tous les sommets de V_1 appartiennent à M. Cela signifie que tous les sommets mobiles seront insérés dans S'. De plus, puisque M est un couplage, chaque sommet est inséré au plus une fois. Donc chaque sommet mobile u est inséré une fois et une seule dans S', et est inséré entre deux sommets fixes (v et $s(v)$, qui sont tout deux des sommets fixes, par construction).

Montrons maintenant que la solution obtenue est réalisable : chaque tour doit comprendre le dépôt et il doit y avoir au plus $C+1$ sommets par tour (C clients et le dépôt). Nous savons que, pour chaque $i \in \{1, \ldots, k\}$, le nombre d'arêtes dans $M \cap E_i$ est inférieur ou égal à $r_i = C - |T_i| + 1$. $|T_i|$ est le nombre de sommets fixes du $i^{\text{ème}}$ tour, en comptant le dépôt. Le nombre d'arêtes dans $M \cap E_i$ correspond au nombre de sommets mobiles ajoutés à ce tour. Le nombre de sommets dans le $i^{\text{ème}}$ tour de S' est donc égal à $|T_i| + |M \cap E_i| \leq |T_i| + C - |T_i| + 1 \leq C + 1$, et le $i^{\text{ème}}$ tour contient le dépôt.

Ainsi, chaque solution du problème de couplage complet contraint par partitions correspond à une solution voisine de S, et chaque solution voisine de S correspond à une solution du problème de couplage complet contraint par partitions. Puisque S' est une solution de coût minimum, S' fait partie des meilleurs solutions voisines de S. □

3.4 Résolution du problème CCCP_{min}

Le problème CCCP de poids minimum peut être défini par le programme linéaire en nombres entiers suivant : Nous avons un graphe biparti $G = (V, E)$ avec un ensemble de sommets $V =$

(V_1, V_2), et une partition $\{T_1, T_2, \ldots, T_k\}$ des sommets de V_2. Soit $\{E_1, E_2, \ldots E_k\}$ une partition des arêtes de E telle que $[u, v] \in E_i$ si et seulement si $v \in T_i$.

$$\text{Minimiser} \sum_{[u,v]\in E} c_{uv} x_{uv} \tag{3.1}$$

sous les contraintes :

$$\sum_{v\in V_2,[u,v]\in E} x_{uv} = 1 \quad \forall u \in V_1 \tag{3.2}$$

$$\sum_{u\in V_1,[u,v]\in E} x_{uv} \leq 1 \quad \forall v \in V_2 \tag{3.3}$$

$$\sum_{[u,v]\in E_i} x_{uv} \leq r_i \quad \forall i \in \{1, \ldots, k\} \tag{3.4}$$

$$x_{uv} \in \{0, 1\} \quad \forall [u, v] \in E. \tag{3.5}$$

Nous avons $x_{vw} = 1$ si l'arête $[v, w]$ appartient au couplage recherché, et $x_{vw} = 0$ dans le cas contraire. Puisque la matrice des contraintes du programme linéaire est totalement unimodulaire (voir [15]), le problème peut être résolu en temps polynomial en utilisant des techniques de programmation linéaire. Il est cependant plus efficace d'utiliser des techniques de flot pour ce problème.

Nous pouvons exprimer le problème CCCP$_{min}$ par le problème de flot suivant : nous avons un graphe orienté $G' = (V', E')$ dans lequel chaque arc (u, v) a un coût $w_{u,v}$ et une capacité $Cap_{u,w}$, nous avons une source et un puits, et nous cherchons un flot maximum de coût minimum allant de la source au puits. Les sommets V' sont les suivants : la source, les sommets de V_1 et V_2 (qui appartiennent aussi à G), k sommets $\{a_1, a_2, \ldots, a_k\}$, et le puits. La source a un arc sortant vers chaque sommet de V_1. Chaque sommet $u \in V_1$ a un lien sortant vers le sommet $v \in V_2$ si et seulement si il existe une arête $[u, v]$ dans E. Chaque sommet $u \in T_i$ a un arc sortant vers le sommet a_i, et les sommets $\{a_1, a_2, \ldots, a_k\}$ ont un arc sortant vers le puits. Le coût de chaque arc est 0, sauf pour les arcs entre les sommets de V_1 et les sommets de V_2. Soit $u \in V_1$ et $v \in V_2$, le coût de l'arc (u, v) est égal au coût de cette arête dans le problème CCCP de poids minimum correspondant : $w_{u,v} = c_{u,v}$. La capacité de chaque arc est 1, sauf pour les arcs entre les sommets $\{a_1, a_2, \ldots, a_k\}$ et le puits : la capacité de l'arc entre a_i et le puits est égale à r_i. $|V_1|$ unités de flot partent de la source.

Exemple 5 : La figure 3.5 montre l'instance de flot correspondant à l'instance du problème CCCP de poids minimum montrée sur la figure 3.2. Les arcs sont orientés de la gauche vers la droite. La capacité de chaque arc est égale à Cap, ou 1 quand Cap n'est pas indiqué, et le coût de chaque arc est égal à 0, sauf pour les arcs entre les sommets de V_1 et les sommets de V_2.

Nous pouvons trouver polynomialement (en un temps en $O(n^3)$, où n est le nombre de sommets) le flot maximum de coût minimum en utilisant un algorithme de flot [3]. Connaissant cette solution, nous pouvons déduire une solution au problème CCCP de poids minimum correspondant : une arête entre un sommet de V_1 et V_2 appartient au couplage complet contraint par partitions et de

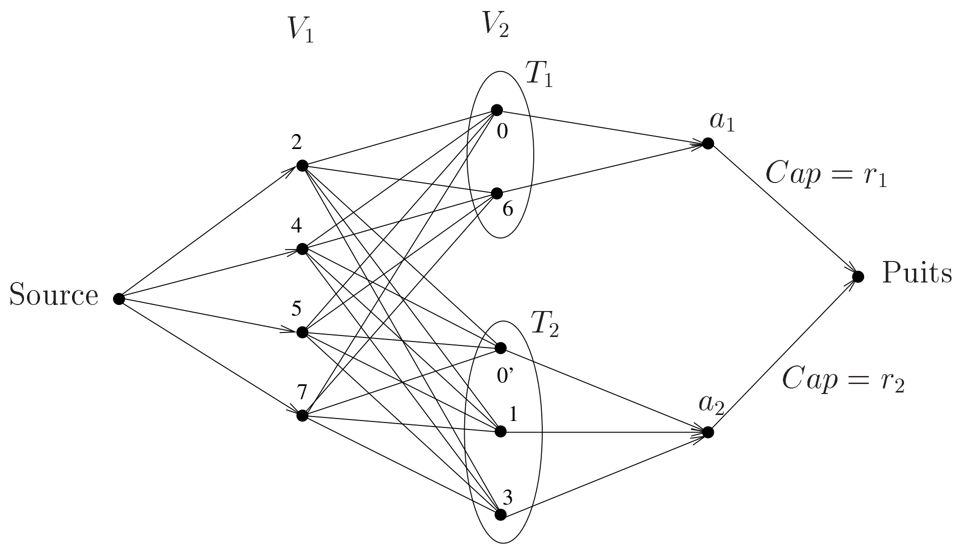


FIG. 3.5 – Instance du problème de flot correspondant à l'instance du problème de couplage de la figure 3.2.

poids minimum si et seulement si il y a une unité de flot qui traverse cette arête dans G'. En effet, de chaque sommet de V_1 part une unité de flot, il y a au plus une unité de flot qui traverse chaque sommet de V_2, et il n'y a pas plus de r_i unités de flot qui viennent des sommets de T_i.

Grâce à la réduction présentée dans la section précédente, nous obtenons le résultat suivant :

Corollaire 3.1 *La recherche de la meilleure solution voisine dans le voisinage exponentiel multi-insertions, pour le problème de tournées de véhicules avec des demandes unitaires, peut être effectuée en temps polynomial.*

Nous pouvons remarquer qu'il est possible d'utiliser ce voisinage dans le cadre d'une recherche tabou, en donnant un coût important à certaines arêtes afin d'empêcher un sommet mobile d'être inséré à la même position que dans la solution initiale.

3.5 Cas dans lequel les clients ont des demandes différentes

Nous avons considéré jusqu'à présent une version restreinte du PTV dans laquelle chaque client demande exactement une unité de bien. Nous pouvons nous demander si le voisinage exponentiel que nous avons étudié peut aussi être exploré en temps polynomial si les demandes des clients sont arbitraires, i.e. si chaque client demande un nombre (entier) d'unités de bien, qui n'est pas nécessairement un. Dans ce cas, le voisinage multi-insertions est celui présenté dans la section 3.2, et dans lequel une solution est réalisable si et seulement si la somme des demandes des clients d'un même tour est inférieure ou égale à la capacité du véhicule.

Théorème 3.3 *Il est NP-difficile de trouver la meilleure solution du voisinage multi-insertions si les demandes des clients sont arbitraires, et ce même si les distances entre les clients sont toutes identiques.*

Preuve : Nous réduisons le problème de PARTITION [40] à notre problème. Étant donnés $n \geq 3$ nombre entiers $x_1, \ldots, x_n$ tels que $\sum_{i=1}^{n} x_i = 2\,L$ pour un entier L , le problème PARTITION consiste à savoir s'il existe un sous-ensemble $B \subseteq \{1, \ldots, n\}$ tel que $\sum_{i \in B} x_i = L$. Étant donnée une instance de PARTITION, nous construisons l'instance suivante de notre problème :

Nous avons un dépôt et $3\,n - 4$ clients. Dans la solution initiale nous avons $n + 2$ tours : n tours de longueur 2, dans lesquels il y a un seul client à servir, et 2 tours dans lesquels il y a $n - 2$ clients par tour. Les sommets mobiles sont les clients des n premiers tours (les tours de longueur 2) ; les autres clients et le dépôt sont des sommets fixes. La figure 3.6 nous montre une telle instance quand $n = 5$. Chaque client fixe a une demande égale à 1, chaque client mobile i a une demande égale à x_i, et la capacité du véhicule est $n - 2 + \frac{\sum_{i=1}^{n} x_i}{2}$. La distance entre chaque couple de sommets est 1.

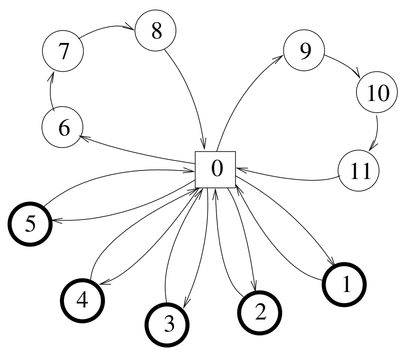


FIG. 3.6 – Les sommets mobiles sont les sommets 1 à 5. Le sommet mobile i a une demande égale à x_i. La meilleure solution voisine a un coût de 13 si et seulement si il existe une partition de $\{x_1, x_2, x_3, x_4, x_5\}$.

Nous devons maintenant insérer les sommets mobiles entre les sommets fixes, et nous souhaitons minimiser la longueur des tours. La meilleure solution est obtenue quand le coût de l'ajout d'un sommet mobile est d'un : la distance totale est alors égale à $3\,n - 2$. Ceci est possible si et seulement si les sommets mobiles ne sont pas insérés dans un tour dans lequel le seul sommet fixe est le dépôt, i.e. ceci est possible si les sommets sont insérés dans un des deux grands tours. Puisque la capacité du véhicule est $n - 2 + \frac{\sum_{i=1}^{n} x_i}{2}$, et puisque chaque client fixe a une demande égale à 1, la somme des demandes des clients mobiles insérés dans un même grand tour doit être inférieure ou égale à $\frac{\sum_{i=1}^{n} x_i}{2}$. La somme totale des demandes des clients mobiles est égale à $\sum_{i=1}^{n} x_i$, donc il est possible d'insérer tous les clients mobiles dans les deux grands tours (et ainsi d'avoir une longueur totale des tours égale à $3\,n - 2$) si et seulement si il existe une partition (M_1, M_2) des clients mobiles telle que la somme des demandes des clients de M_1 est égale à la somme des demandes des clients de M_2, c'est-à-dire s'il existe une solution au problème de PARTITION. □

3.6 Conclusion

Nous avons présenté un voisinage de taille exponentielle pour le PTV qui peut être exploré en temps polynomial en utilisant des techniques de flot. Le voisinage que nous avons considéré est une généralisation de celui proposé par Gutin [44] pour le problème du voyageur de commerce. Étant donnée la relation entre les deux problèmes, il était naturel de regarder d'un point de vue théorique si l'exploration de ce voisinage pour le PTV était aussi polynomiale.

Nous avons montré que l'exploration du voisinage multi-insertions est polynomiale si chaque client demande la même quantité de bien, et cela quelle que soit la configuration des distances entre les clients (matrices des distances symétrique ou pas, respectant ou non l'inégalité triangulaire). Dans le cas où les clients demandent des quantités de bien différentes, le problème devient NP-difficile, et cela même si toutes les distances (entre chaque couple de clients, et entre le dépôt et chaque client) sont égales.

D'un point de vue plus pratique, une question intéressante serait d'étudier, pour le voisinage multi-insertions que nous avons proposé, l'impact du choix des sommets mobiles/fixes sur la qualité de l'optimum local trouvé ensuite. Plus généralement, il serait aussi intéressant de comparer expérimentalement ce voisinage avec d'autres voisinages.

Enfin, nous remarquons que, alors que le problème de couplage complet contraint dans un graphe biparti (dans lequel le couplage peut contenir au plus r_i sommets de l'ensemble d'arêtes E_i) est NP-difficile [52] dans le cas général, et qu'il est polynomial dans le cas où il y a une seule contrainte (i.e. il y a un seul ensemble E_1 dans lequel on peut choisir au plus r_1 arêtes), ce problème peut être résolu en temps polynomial si les ensembles d'arêtes $\{E_i\}$ correspondent à une partition de V_2 (et ce, même si les arêtes sont pondérées et que l'on recherche un couplage de poids minimum).

Chapitre 4

Ordonnancements et équité

Nous nous intéressons dans ce chapitre à des mesures d'équité dans un problème d'ordonnancement : équité entre les machines (nous voulons minimiser le nombre moyen de tâches non terminées par machine) et équité entre les tâches (nous supposons que le but de chaque tâche est de se terminer le plus rapidement possible). Nous nous intéressons plus particulièrement à ces mesures d'équité parmi l'ensemble des ordonnancements qui minimisent la date de fin moyenne des tâches. Ces travaux ont été présentés dans [8, 9] et publiés dans [6].

4.1 Introduction

Nous supposons que nous devons ordonnancer n tâches numérotées de 1 à n et de longueurs $l_1, \ldots, l_n$ sur m machines $P_1, \ldots, P_m$. Étant donné un ordonnancement, nous notons C_j la date de fin de la tâche j, et X (respectivement $\overrightarrow{X}$) le vecteur des dates de fin dont la $i^{\text{ème}}$ coordonnée est C_i pour tout $i \in \{1, \ldots, n\}$ (respectivement où les coordonnées de X ont été réordonnées par ordre croissant). Une instance consiste en un nombre m de machines et un ensemble de tâches. Pour chaque instance $\mathcal{I}$, $V(\mathcal{I})$ est l'ensemble des vecteurs (des dates de fin) induits par des ordonnancements réalisables de l'instance $\mathcal{I}$.

Un des critères les plus populaires dans la théorie de l'ordonnancement est le critère de la *somme des dates de fin*, défini comme $\sum_{i=1}^{n} C_i$, et que l'on souhaite minimiser. Nous noterons par la suite ce problème Min SDF. Ce problème est notamment équivalent à la minimisation de la date de fin moyenne des tâches. Une solution optimale de Min SDF peut être obtenue en utilisant l'algorithme de liste SPT. Nous rappelons ici l'algorithme SPT : ordonner les tâches par ordre de longueurs croissantes puis les ordonnancer de manière gloutonne sur les machines (quand une machine devient inactive, elle exécute la plus petite tâche non encore ordonnancée). R. Conway, W. Maxwell, et L. Miller ont défini dans [33] la classe des ordonnancements optimaux pour le problème Min SDF. Cette classe, plus large que le seul ordonnancement SPT décrit ci-dessus, est définie en utilisant la notion de rang, que nous rappelons maintenant. Nous supposons que nous avons n tâches, numérotées de façon à ce que $l_1 \leq l_2 \leq \cdots \leq l_n$, à ordonnancer sur m machines. Soit π_i un ensemble de longueurs défini comme suit : $\pi_k = \{l_n, l_{n-1}, \ldots, l_{n-m+1}\}$, $\pi_{k-1} = \{l_{n-m}, \ldots, l_{n-2m+1}\}, \ldots, \pi_1 = \{l_{n-(k-1)m}, \ldots, l_1\}$, où $k = \lceil \frac{n}{m} \rceil$. L'ensemble π_i est appelé le $i^{\text{ème}}$*rang* des tâches. Considérons maintenant un ordonnancement obtenu en ordonnançant les tâches rang par rang, dans l'ordre $\pi_1, \pi_2, \ldots, \pi_k$ et tel qu'il n'y ait pas deux tâches de même rang sur la même machine : les tâches de π_1 sont ordonnancées en première position, chacune sur une machine, puis on ordonnance les tâches de π_2, chacune étant également sur une machine différente, et ainsi de suite. Quelle que soit l'affectation des tâches d'un même rang sur les différentes machines, l'ordonnancement retourné est optimal pour le problème $\textsc{Min SDF}$. Cette famille d'ordonnancements sera appelée par la suite la famille des *ordonnancements SPT*. Notons que cette dénomination, introduite dans [33], n'est valable que pour ce chapitre, un ordonnancement SPT correspondant dans les autres chapitres à l'algorithme glouton présenté au début de ce chapitre (et dans le chapitre 1), et que nous appellerons dans ce chapitre $\mathrm{SPT}_{\mathrm{glouton}}$.

Les ordonnancements SPT sont les seuls ordonnancements qui sont optimaux pour la somme des dates de fin (et donc pour la date de fin moyenne) [33]. Nous nous intéressons donc à étudier les performances de ces ordonnancements pour d'autres critères d'optimalité, pouvant être considérés comme des critères d'équité, et qui sont les suivants :

- **la somme maximum des dates de fin par machine** : $\max_{1 \leq i \leq m} \sum_{j \text{ sur } P_i} C_j$. Cette mesure capture le souhait de distribuer autant que possible la somme totale des dates de fin parmi les m machines du système. La somme des dates de fin représente le nombre moyen (sur une durée supérieure ou égale à la durée de l'ordonnancement) de tâches en attente (i.e. non encore terminées) par unité de temps sur cette machine. La somme des dates de fin d'une machine divisée par la date de fin de l'ordonnancement indique ainsi le nombre moyen de tâches en attente par unité de temps sur cette machine pendant l'ordonnancement. La somme *maximum* des dates de fin par machine indique donc le nombre moyen maximum de tâches en attente sur une machine au cours de la durée de l'ordonnancement. Si toutes les tâches occupent la même place en mémoire (la longueur d'une tâche représentant sa durée d'exécution), et si les tâches sont supprimées de la mémoire de la machine une fois qu'elles ont été exécutées, alors la somme des dates de fin par machine représente la mémoire occupée en moyenne par machine, et le problème $\textsc{Min Max SDF}$ cherche donc à minimiser le maximum de mémoire occupée en moyenne par machine.
- **la mesure d'équité globale**, appelée “global fairness” en anglais et introduite par A. Kumar et J. Kleinberg dans [57] : soit une instance $\mathcal{I}$ et deux vecteurs de dates de fin $X, Y \in V(\mathcal{I})$, nous écrirons $X \preccurlyeq Y$ si et seulement si $X_i \leq Y_i$ pour tout $i \in \{1, \ldots, n\}$, où X_i (respectivement Y_i) représente la $i^{\text{ème}}$ coordonnée de X (respectivement Y). Le *rapport d'approximation global* de X, noté $c_{gbl}(X)$, est égal au plus petit α tel que $\overrightarrow{X} \preccurlyeq \alpha\overrightarrow{Y}$ pour tout $Y \in V(\mathcal{I})$. Ainsi $c_{gbl}(X)$ est le plus petit nombre α pour lequel le vecteur des dates de fin X dans lequel les valeurs ont été réordonnées par ordre croissant est α-approché de n'importe quel autre vecteur des dates de fin $Y \in V(\mathcal{I})$ dans lequel les valeurs ont aussi été réordonnées. Le meilleur rapport d'approximation global pour l'instance $\mathcal{I}$ est le suivant :

$$c^*_{gbl}(\mathcal{I}) = \min_{X \in V(\mathcal{I})} c_{gbl}(X).$$

- **la mesure d'équité individuelle**, aussi appelé mesure de *satisfaction individuelle* : le *rapport de satisfaction individuelle* compare la date de fin de chaque tâche avec la plus petite date de fin possible que cette tâche aurait pu avoir dans un autre ordonnancement réalisable de la même instance. Soit $X \in \mathcal{I}$. Le rapport de satisfaction individuel de X, $c_{ind}(X)$, est égal au plus petit α tel que $X \preccurlyeq \alpha Y$ pour tout $Y \in V(\mathcal{I})$. Le meilleur rapport de satisfaction individuel possible pour l'instance $\mathcal{I}$ est le suivant :

$$c^*_{ind}(\mathcal{I}) = \min_{X \in V(\mathcal{I})} c_{ind}(X).$$

Notre approche est similaire à celle de J. Bruno, E. Coffman et R. Sethi qui ont considéré dans [27] la question suivante : parmi tous les ordonnancements optimaux pour le critère de la *somme des dates de fin*, est-il possible de trouver celui qui minimise la *date de fin de l'ordonnancement* ? Les auteurs ont montré que ce problème est en fait NP-difficile. C. Stein et J. Wein étudient également dans [76] un problème proche.

Plan du chapitre. Dans la section 4.2 nous considérons le problème qui consiste à *minimiser la somme maximum des dates de fin par machine* (Min Max SDF). Nous montrons tout d'abord que, contrairement à la minimisation de la somme des dates de fin, le problème Min Max SDF est NP-difficile. De plus nous montrons que l'algorithme de liste classique $\mathrm{SPT}_{\mathrm{glouton}}$ a un rapport d'approximation inférieur ou égal à $(3 - \frac{3}{m} + \frac{1}{m^2})$ pour ce problème, et qu'il existe des instances pour lesquelles aucun ordonnancement $\mathrm{SPT}_{\mathrm{glouton}}$ ne peut avoir un rapport d'approximation meilleur que $2 - \frac{2}{m^2+m}$. Dans la section 4.3, nous considérons le *rapport d'approximation global* et nous montrons que les ordonnancements $\mathrm{SPT}_{\mathrm{glouton}}$ ont un rapport d'approximation global égal à $2 - \frac{1}{m}$ et qu'aucun algorithme ne peut avoir un meilleur rapport d'approximation global. Notons que C. Philips, C. Stein et J. Wein [68] ont donné un algorithme 3-approché pour le même problème quand des dates d'apparition ("release dates" en anglais) sont prises en compte. Pour le *rapport de satisfaction individuelle* cependant le rapport ne peut pas être borné par une constante mais nous montrons qu'un ordonnancement $\mathrm{SPT}_{\mathrm{glouton}}$ obtient le meilleur rapport possible. Finalement, nous nous intéressons à une version plus restreinte du rapport de satisfaction individuelle où une solution est comparée uniquement aux autres solutions de la famille SPT, et nous montrons que dans ce cas le rapport de satisfaction individuelle est égal à 2.

4.2 Le problème Min Max SDF

Nous rappelons que Min SDF est le problème qui consiste à *minimiser la somme des dates de fins* de toutes les tâches, $\sum_{i=1}^{n} C_j$. Nous remarquons que le problème Min SDF et le problème Min Max SDF sont différents. Ceci est facile à voir en considérant l'instance suivante : trois tâches de longueur 1 et une tâche de longueur 5 sur deux machines identiques (voir figure 4.1).

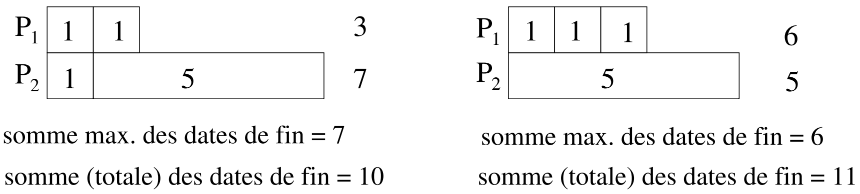


Fig. 4.1 – Instance pour laquelle les solutions optimales des problèmes Min SDF et Min Max SDF sont différentes.

Dans la solution optimale pour le problème MIN SDF, deux tâches de taille 1 sont sur une même machine, et les autres tâches sont sur l'autre machine (la somme totale des dates de fin est égale à 10, et la somme maximale des dates de fin par machine est égale à 7). Dans la solution optimale du problème MIN MAX SDF, les trois tâches de longueur 1 sont sur la même machine, et la tâche de longueur 5 est sur l'autre machine (la somme totale des dates de fin est égale à 11, et la somme maximale des dates de fin par machine est égale à 6).

Nous avons vu dans la section précédente que le problème MIN MAX SDF minimise le maximum, parmi les machines, du nombre moyen (sur un intervalle de temps comprenant la période entre la date de début et la date de fin de l'ordonnancement) de tâches en attente sur une même machine. Considérons par exemple la figure 4.1 à droite. La date de fin de cet ordonnancement est 5. Il y a sur la machine P_2 une seule tâche, de longueur 5, qui s'exécute entre la date 0 et la date 5. Sur cette machine, le nombre moyen de tâches non terminées est donc de 1 pendant cette période (puisqu'il y a, pour chaque unité de temps entre les dates 0 et 5, exactement une tâche non terminée sur cette machine). Il y a trois tâches de longueur 2 sur la machine P_1, celles-ci se terminant aux dates 1, 2, et 3. Pendant la première unité de temps (entre les dates 0 et 1), il y a donc 3 tâches attendant la fin de leur exécution. Pendant la deuxième unité de temps, il y a 2 tâches en attente, et lors de la troisième unité de temps il ne reste plus qu'une tâche attendant la fin de son exécution (pendant les quatrièmes et cinquièmes unités de temps, il n'y a plus de tâches en attente). Le nombre moyen de tâches attendant la fin de leur exécution au cours des 5 unités de temps de l'ordonnancement sur la machine P_1 est donc égal à $\frac{3+2+1}{5} = 1,2$.

4.2.1 Complexité du problème

Nous prouvons dans cette section que le problème MIN MAX SDF est NP-difficile.

Théorème 4.1 *Le problème* MIN MAX SDF *est NP-difficile.*

Preuve : Considérons la version de décision du problème MIN MAX SDF :

Problème MIN MAX SDF : Étant donné un ensemble de tâches à ordonnancer, un nombre m de machines, et un nombre k, existe-t-il un ordonnancement tel que la somme maximale des dates de fin par machine est inférieure ou égale à k ?

Nous montrons que le problème PARTITION, qui est NP-difficile [40], peut s'exprimer sous la forme du problème MIN MAX SDF. Nous avons déjà rencontré ce problème dans le chapitre précédent, nous en rappelons sa définition ici :

Problème de PARTITION : Étant donné un ensemble $C = \{x_1, x_2, \ldots, x_n\}$ de nombres entiers, existe-t-il une partition (A, B) de C, i.e. $A \cup B = C$ et $A \cap B = \emptyset$, telle que $\sum_{x \in A} x = \sum_{x \in B} x$?

Étant donnée une instance I_1 de PARTITION : un ensemble $C = \{x_1, x_2, \ldots, x_n\}$ de nombres tels que $x_1 \leq x_2 \leq \cdots \leq x_n$, nous construisons (en un temps polynomial) une instance I_2 du problème MIN MAX SDF telle que la réponse (affirmative ou négative) du problème MIN MAX SDF sur l'instance I_2 est égale à la réponse du problème PARTITION sur l'instance I_2. Cette instance I_2 est la suivante : $k = \frac{3}{2}(\sum_{i=1}^{n} x_i) - \sum_{i=1}^{n} \frac{x_i}{n-i+1}$, et nous avons $2n$ tâches $\{1, 2, \ldots, 2n\}$ à

ordonnancer sur 2 machines P_1 et P_2. Nous utilisons la notation habituelle l_i pour noter la longueur de la tâche i. Les longueurs des tâches sont telles que : $l_{2j-1} = \sum_{i=1}^{j-1} \frac{x_i}{n-i+1}$ et $l_{2j} = \sum_{i=1}^{j} \frac{x_i}{n-i+1}$, pour tout $j \in \{1, 2, \ldots, n\}$.
Par exemple, (si $n \geq 6$), $l_1 = 0,\ l_2 = \frac{x_1}{n}$,

$$l_3 = \frac{x_1}{n},\ l_4 = \frac{x_1}{n} + \frac{x_2}{n-1},$$
$$l_5 = \frac{x_1}{n} + \frac{x_2}{n-1},\ l_6 = \frac{x_1}{n} + \frac{x_2}{n-1} + \frac{x_3}{n-2},$$
$$l_{2n-1} = \frac{x_1}{n} + \frac{x_2}{n-1} + \cdots + \frac{x_{n-1}}{2},$$
$$l_{2n} = \frac{x_1}{n} + \frac{x_2}{n-1} + \cdots + \frac{x_{n-1}}{2} + x_n$$

La transformation de l'instance I_1 en l'instance I_2 se fait en un temps polynomial. Montrons maintenant que la solution du problème MIN MAX SDF sur l'instance I_2 est positive si et seulement si la solution du problème PARTITION sur l'instance I_1 est positive.

Nous montrons tout d'abord que si la réponse au problème PARTITION sur l'instance I_1 est positive alors la réponse au problème MIN MAX SDF sur l'instance I_2 est également positive : s'il existe une partition (A, B) de C alors il existe un ordonnancement des tâches tel que la somme maximale des dates de fin est égale à $\frac{3}{2}(\sum_{i=1}^{n} x_i) - \sum_{i=1}^{n} \frac{x_i}{n-i+1}$. Nous pouvons obtenir un tel ordonnancement en affectant, pour tout $j \in \{1, \ldots, n\}$, les tâches $(2j-1)$ et $2j$ au rang j : on affecte la tâche $2j$ à P_1 et la tâche $(2j-1)$ à P_2 si $x_j \in A$, sinon on affecte la tâche $2j$ à P_2 et la tâche $(2j-1)$ à P_1.
En effet, la longueur d'une tâche ajoutée au $j^{\text{ème}}$ rang sera comptée $n-j+1$ fois dans la somme des dates de fin de sa machine. Par exemple, la longueur de la dernière tâche d'une machine sera comptée seulement une fois, alors que la longueur de la première tâche d'une machine sera comptée n fois (car il y a n tâches sur chaque machine). Ainsi, la contribution à la somme des dates de fin par machine de la tâche $(2j-1)$ (respectivement $2j$), placée au $j^{\text{ème}}$ rang, est égale à $(n-j+1) \times l_{2j-1}$ (respectivement $(n-j+1) \times l_{2j} = (n-j+1) \times (l_{2j-1} + \frac{x_j}{n-j+1})$). La différence entre ces deux contributions est égale à x_j : si $x_j \in A$ (i.e. la tâche $2j$ est placée sur P_1) alors la contribution de la $j^{\text{ème}}$ tâche de P_1 est égale à la contribution de la $j^{\text{ème}}$ tâche de P_2, plus x_j. De la même manière, si $x_j \in B$ (i.e. la tâche $2j$ est placée sur P_2) alors la contribution de la $j^{\text{ème}}$ tâche de P_2 est égale à la contribution de la $j^{\text{ème}}$ tâche de P_1, plus x_j.
La somme des dates de fin de P_1 est donc $\sum_{j=1}^{n} l_{2j-1} + \sum_{x_j \in A} x_j$, et la somme des dates de fin de P_2 est $\sum_{j=1}^{n} l_{2j-1} + \sum_{x_j \in B} x_j$. De plus $\sum_{j=1}^{n} l_{2j-1} = (n-1)\frac{x_1}{n} + (n-2)\frac{x_2}{n-1} + \cdots + \frac{x_{n-1}}{2} = x_1 - \frac{x_1}{n} + x_2 - \frac{x_2}{n-1} + \cdots + x_{n-1} - \frac{x_{n-1}}{2} = \sum_{i=1}^{n} x_i - \sum_{i=1}^{n} \frac{x_i}{n-i+1}$. La somme des dates de fin des tâches de P_1 est donc $(\sum_{i=1}^{n} x_i - \sum_{i=1}^{n} \frac{x_i}{n-i+1}) + \sum_{x_j \in A} x_j$, et la somme des dates de fin des tâches de P_2 est $(\sum_{i=1}^{n} x_i - \sum_{i=1}^{n} \frac{x_i}{n-i+1}) + \sum_{x_j \in B} x_j$.
S'il existe une partition de l'instance I_1 (i.e. $\sum_{x_j \in A} = \sum_{x_j \in B} = \frac{\sum_{i=1}^{n} x_i}{2}$) alors la somme maximale des dates de fin par machine de l'instance I_2 est égale à $(\sum_{i=1}^{n} x_i - \sum_{i=1}^{n} \frac{x_i}{n-i+1}) + \frac{\sum_{i=1}^{n} x_i}{2} = \frac{3}{2}(\sum_{i=1}^{n} x_i) - \sum_{i=1}^{n} \frac{x_i}{n-i+1} = k$. Ainsi si la réponse au problème PARTITION sur l'instance I_1 est positive alors la réponse au problème MIN MAX SDF sur l'instance I_2 est également positive.

Nous montrons maintenant que si la réponse au problème MIN MAX SDF sur l'instance I_2 est positive, alors la réponse au problème PARTITION sur l'instance I_1 sera également positive. Si la somme maximale des dates de fin par machine est inférieure ou égale à $\frac{3}{2}(\sum_{i=1}^{n} x_i) - \sum_{i=1}^{n} \frac{x_i}{n-i+1}$, alors nous avons un ordonnancement SPT. En effet, la somme totale des dates de fin d'un ordonnancement SPT est $S = 2 \times \left(\frac{3}{2}(\sum_{i=1}^{n} x_i) - \sum_{i=1}^{n} \frac{x_i}{n-i+1}\right)$, comme nous l'avons vu ci-dessus, et un ordonnancement a une somme des dates de fin minimale si et seulement si c'est un ordonnancement SPT [27]. Puisque la somme totale des dates de fin ne peut pas être supérieure à deux fois la somme maximale des dates de fin par machine, un ordonnancement qui est une solution au problème MIN MAX SDF est un ordonnancement SPT. Dans n'importe quel ordonnancement SPT, les tâches $2i$ et $2i-1$ sont au $i^{\text{ème}}$ rang, car $l_1 < l_2 \leq l_3 < \cdots \leq l_{2j-1} < l_{2j} \leq l_{2j+1} < l_{2j+2} \leq \cdots < l_n$. Donc, dans n'importe quel ordonnancement SPT, la somme des dates de fin des tâches de P_1 moins celle des tâches de P_2 est égale à $\sum_{2j \text{ sur } P_1} x_j - \sum_{2j \text{ sur } P_2} x_j$. S'il existe une solution au problème MIN MAX SDF avec $k = \frac{3}{2}(\sum_{i=1}^{n} x_i) - \sum_{i=1}^{n} \frac{x_i}{n-i+1}$ alors la somme des dates de fin de P_1 est égale à la somme des dates de fin de P_2 (car sinon la somme totale des dates de fin des tâches serait inférieure à S), et alors $\sum_{2j \text{ sur } P_1} x_j = \sum_{2j \text{ sur } P_2} x_j$: il existe donc une partition de C et nous pouvons construire cette partition (A, B) en plaçant x_j dans A si la tâche $2j$ est sur P_1 et x_j dans B si la tâche $2j$ est sur P_2. □

4.2.2 Approximation

Dans cette section nous montrons qu'un algorithme optimal pour la minimisation de la somme des dates de fin retourne des solutions 3-approchées pour le problème MIN MAX SDF. Considérons l'algorithme suivant, noté $\mathrm{SPT}_{\mathrm{glouton}}$:

> Ordonner les tâches par longueurs non décroissantes. À chaque étape i, pour $1 \leq i \leq n$, ordonnancer la tâche courante sur la machine qui a la plus petite date de fin.

Cet algorithme retourne une solution optimale au problème MIN SDF, qui consiste à minimiser la somme des dates de fin des tâches. Montrons que cet algorithme est $(3 - \frac{3}{m} + \frac{1}{m^2})$-approché pour le problème MIN MAX SDF.

Afin de rendre $\mathrm{SPT}_{\mathrm{glouton}}$ déterministe quand plusieurs machines ont la même date de fin à une étape donnée, et afin d'analyser plus facilement cet algorithme, nous nous référerons dans les preuves à l'algorithme suivant, qui est un algorithme SPT glouton (le lemme 4.1 montre qu'il ajoute à chaque étape une tâche sur la machine qui a la plus petite date de fin) :

> Ordonner les tâches par longueurs non décroissantes. À chaque étape i, pour $1 \leq i \leq n$, ordonnancer la tâche courante sur la machine $P_{i \bmod m}$.

Lemme 4.1

a) Au début de l'étape i de $SPT_{glouton}$, la machine $P_{i \bmod m}$ est celle qui a la plus petite date de fin et la plus petite somme des dates de fin.

b) À la fin de l'étape i de $SPT_{glouton}$, la machine $P_{i \bmod m}$ est celle qui a la plus grande somme des dates de fin.

Preuve :

a) Considérons que nous sommes au *début* de l'étape i (i.e. la tâche courante n'a pas encore été ordonnancée sur $P_{i \bmod m}$). Soit $P_{courant}$ la machine $P_{i \bmod m}$, et soit $f_{courant}$ sa date de fin. Soit $n(P_{courant})$ le nombre de tâches sur $P_{courant}$. Montrons que $P_{courant}$ a la plus petite date de fin :

– Pour tout $k > i \bmod m$, la machine P_k a une date de fin supérieure ou égale à $f_{courant}$. En effet, P_k a le même nombre de tâches que $P_{courant}$ et, pour chaque i tel que $1 \leq i \leq n(P_{courant})$, la longueur de la $i^{\text{ème}}$ tâche de P_k est supérieure ou égale à la longueur de la $i^{\text{ème}}$ tâche de $P_{courant}$, par construction.

– Pour tout $k < i \bmod m$, la machine P_k a une date de fin supérieure ou égale à $f_{courant}$. En effet, le nombre de tâches sur P_k est égal à $n(P_{courant}) + 1$ et, pour tout i tel que $1 \leq i \leq n(P_{courant})$, la longueur de la $(i+1)^{\text{ème}}$ tâche de P_k est supérieure ou égale à la longueur de la $i^{\text{ème}}$ tâche de $P_{courant}$, par construction.

Le raisonnement est le même pour montrer que $P_{courant}$ a la plus petite somme des dates de fin.

b) Considérons que nous sommes à la *fin* de l'étape i. Soit $P_{courant}$ la machine $P_{i \bmod m}$, et soit $s_{courant}$ la somme des dates de fin de tâches ordonnancées sur $P_{courant}$ à cette étape. Soit $n(P_{courant})$ le nombre de tâches sur $P_{courant}$. Montrons que $P_{courant}$ a la plus grande somme des dates de fin :

– Pour tout $k > i \bmod m$, la machine P_k a une somme des dates de fin inférieure ou égale à $s_{courant}$. En effet, P_k a une tâche de moins que $P_{courant}$ et, pour chaque i tel que $1 \leq i < n(P_{courant})$, la longueur de la $i^{\text{ème}}$ tâche de P_k est inférieure ou égale à la longueur de la $(i+1)^{\text{ème}}$ tâche de $P_{courant}$, par construction.

– Pour tout $k < i \bmod m$, la machine P_k a une somme des dates de fin inférieure ou égale à $s_{courant}$. En effet, P_k a le même nombre de tâches que $P_{courant}$ et, pour chaque i tel que $1 \leq i \leq n(P_{courant})$, la longueur de la $i^{\text{ème}}$ tâche de P_k est inférieure ou égale à la longueur de la $i^{\text{ème}}$ tâche de $P_{courant}$, par construction.

$\square$

Lemme 4.2 *Soit OPT la somme maximum des dates de fin par machine dans une solution de* MIN MAX SDF. *Nous avons :*

$$OPT \geq \frac{Min(\sum_{j=1}^{n} C_j)}{m},$$

où $Min(\sum_{j=1}^{n} C_j)$ est la somme des dates de fin dans une solution optimale du problème MIN SDF.

Preuve : Dans une solution optimale de MIN MAX SDF chaque machine a une somme des dates de fin inférieure ou égale à OPT : $\forall i \in \{1, \ldots, m\},\ \sum_{j \text{ sur } P_i} C_j \leq OPT$.

Donc $\sum_{j=1}^{n} C_j = \sum_{i=1}^{m} \sum_{j \text{ sur} P_i} C_j \leq m \times OPT$, et $OPT \geq \frac{\sum_{j=1}^{n} C_j}{m} \geq \frac{\text{Min}(\sum_{j=1}^{n} C_j)}{m}$. $\square$

Théorème 4.2 *L'algorithme $SPT_{glouton}$ est $(3 - \frac{3}{m} + \frac{1}{m^2})$-approché pour le problème* MIN MAX SDF.

Preuve : Soit P_x la machine sur laquelle la dernière tâche (la tâche n) est ordonnancée. Avant d'ajouter la tâche n (i.e. quand seulement les $n-1$ premières tâches ont été ordonnancées), P_x est la machine ayant la plus petite somme des dates de fin, notée s, et ayant la plus petite date de fin, notée f (voir lemme 4.1 a). Nous rappelons que nous notons l_n la longueur de la tâche n. La machine sur laquelle la dernière tâche est ordonnancée a la plus grande somme des dates de fin (voir lemme 4.1 b), donc :

$$\begin{array}{rcl} \max_{1\le i\le m}\sum_{j \text{ sur } P_i} C_j & = & \sum_{j \text{ sur } P_x} C_j = s+f+l_n \\ & \le & \frac{\sum_{j=1}^{n-1} C_j}{m} + f + l_n \end{array}$$

Puisque $\text{SPT}_{\text{glouton}}$ retourne une solution optimale pour le problème Min SDF, et puisque la date de fin de la tâche n dans l'ordonnancement retourné par $\text{SPT}_{\text{glouton}}$ est $f+l_n$, nous avons :

$$\begin{array}{rcl} \max_{1\le i\le m}\sum_{j \text{ sur } P_i} C_j & \le & \frac{\text{Min}(\sum_{j=1}^{n} C_j)-(f+l_n)}{m} + f + l_n \\ & \le & \frac{\text{Min}\sum_{j=1}^{n} C_j}{m} + (1-\frac{1}{m})(f+l_n). \end{array}$$

Puisque $f+\frac{l_n}{m} \le OPT$ (car f est la date de fin minimum à l'étape $n-1$), et $l_n \le OPT$, nous avons :

$$\begin{array}{rcl} f+l_n & = & f+\frac{l_n}{m}+\frac{(m-1)\,l_n}{m} \\ & \le & (1+\frac{m-1}{m})OPT \\ & \le & (2-\frac{1}{m})OPT \end{array}$$

De plus, puisque $\frac{\text{Min}\sum_{j=1}^{n} C_j}{m} \le OPT$ (voir lemme 4.2), nous avons :

$$\begin{array}{rcl} \max_{1\le i\le m}\sum_{j \text{ sur } P_i} C_j & \le & OPT + (1-\frac{1}{m})(2-\frac{1}{m})OPT \\ & \le & (3-\frac{3}{m}+\frac{1}{m^2})OPT. \end{array}$$

□

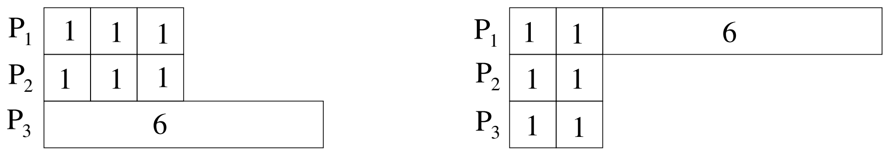
4.2.2.1 Borne inférieure sur le rapport d'approximation de $\text{SPT}_{\text{glouton}}$

Nous montrons maintenant que le rapport d'approximation de $\text{SPT}_{\text{glouton}}$ est supérieur ou égal à $2-\frac{2}{m^2+m}$. Considérons pour cela l'instance suivante : m machines, $m\times(m-1)$ tâches de longueur 1 et une tâche de longueur $B=\sum_{i=1}^{m} i = \frac{m(m+1)}{2}$.

$\text{SPT}_{\text{glouton}}$ ordonnance $m-1$ tâches de longueur 1 sur chaque machine, et la tâche de longueur B sera la dernière tâche de la première machine (voir figure 4.2). La somme maximale des dates de fin est alors $\sum_{i=1}^{m-1} i + (m-1) + B$, ce qui est égal à $2(\sum_{i=1}^{m} i) - 1$.

Une solution optimale au problème Min Max SDF serait la suivante : m tâches de longueur 1 sont ordonnancées sur chacune des $(m-1)$ premières machines, et la tâche de longueur B est sur la dernière machine. La somme maximale des dates de fin dans cette solution est alors égale à $\sum_{i=1}^{m} i$.

Le rapport entre les sommes maximales des dates de fin de ces deux ordonnancements est $\frac{2(\sum_{i=1}^{m} i)-1}{\sum_{i=1}^{m} i}$, ce qui est égal à $2(1-\frac{1}{m^2+m})$.



somme max. des dates de fin = 6 somme max. des dates de fin = 11

FIG. 4.2 – le rapport d'approximation de $\mathrm{SPT}_{\mathrm{glouton}}$ est supérieur ou égal à $(2 - \frac{2}{m^2+m})$: exemple avec $m = 3$.

4.3 Mesures d'équité

Dans cette section, nous considérerons des mesures d'équité afin notamment de comparer l'ordonnancement retourné par l'algorithme glouton SPT, $\mathrm{SPT}_{\mathrm{glouton}}$, à tout autre ordonnancement. Nous appelons $(P \parallel all)$ le problème qui consiste à minimiser la date de fin de chacune des tâches.

4.3.1 Mesure d'équité globale

Nous considérons tout d'abord la mesure d'équité introduite par A. Kumar et J. Kleinberg dans [57], et appelée *rapport d'approximation global* ("global approximation ratio" en anglais). Soit $\mathcal{I}$ une instance de notre problème (un ensemble de tâches $1, \ldots, n$, et un nombre m de machines) et $V(\mathcal{I})$ l'ensemble des vecteurs des dates de fin possibles : un vecteur $X = (x_1, \ldots, x_n)$ appartient à $V(\mathcal{I})$ si et seulement si il existe un ordonnancement réalisable dans lequel, pour tout $i \in \{1, \ldots, n\}$ la tâche i se termine à la date x_i. Soit $X = (x_1, \ldots, x_n)$ un vecteur des dates de fin. $\overrightarrow{X} = (x'_1, \ldots, x'_n)$ représente alors le vecteur X dans lequel les coordonnées on été ordonnées par ordre croissant : pour tout $i \in \{1, \ldots, n\}$, $x'_i \in X$ et $x'_1 \leq x'_2 \leq \cdots \leq x'_n$. Comme nous l'avons indiqué au début de ce chapitre, $c_{gbl}(X)$ est égal au plus petit α pour lequel $\overrightarrow{X}$ est α-approché de tout vecteur $\overrightarrow{Y}$ tel que $Y \in V(\mathcal{I})$. Enfin, le meilleur rapport d'approximation de cette instance est donc $c^*_{gbl}(\mathcal{I}) = \min_{X \in V(\mathcal{I})} c_{gbl}(X)$.

Nous montrons que $\mathrm{SPT}_{\mathrm{glouton}}$ a un rapport d'approximation global égal à $2 - \frac{1}{m}$ pour le problème $(P \parallel all)$, et qu'il n'existe pas d'algorithme ayant un meilleur rapport d'approximation global.

Théorème 4.3 *Soit $\mathcal{I}$ une instance de n tâches à ordonnancer sur m machines. Le rapport d'approximation global du problème $(P \parallel all)$ est égal à $c^*_{gbl}(\mathcal{I}) \leq 2 - \frac{1}{m}$. De plus, le vecteur des dates de fin X de l'ordonnancement retourné par $SPT_{glouton}$ vérifie $c_{gbl}(X) \leq 2 - \frac{1}{m}$.*

Preuve : Considérons une instance $\mathcal{I}$ dans laquelle les tâches $1, \ldots, n$ sont ordonnées par longueurs croissantes. Soit RM le rapport maximal entre l'ordonnancement X_{SPT} retourné par $\mathrm{SPT}_{\mathrm{glouton}}$ et tout autre ordonnancement Y de mêmes tâches : $\mathrm{RM} = \max_Y \frac{\overrightarrow{X_{SPT}}}{\overrightarrow{Y}}$, où $\frac{X}{Y}$ signifie $\max_i \frac{X_i}{Y_i}$. Nous avons donc $\forall\ Y \in V(\mathcal{I})$, $\frac{\overrightarrow{X_{SPT}}}{\overrightarrow{Y}} \leq \mathrm{RM}$, et alors $\overrightarrow{X_{SPT}} \preccurlyeq \mathrm{RM}\ \overrightarrow{Y}$. Donc $\mathrm{RM} = c_{gbl}(X_{SPT})$. Montrons maintenant que RM ne peut pas être supérieur à $2 - \frac{1}{m}$. Considérons sans perte de généralité que ce rapport est égal à la $i^{\text{ème}}$ coordonnée de $\overrightarrow{X_{SPT}}$, divisée par la $i^{\text{ème}}$ coordonnée de $\overrightarrow{Y}$ $(i \in \{1, \ldots, n\})$.

La rapport le pire peut être atteint quand la date de fin de la $i^{\text{ème}}$ tâche est la plus grande possible dans l'ordonnancement $\text{SPT}_{\text{glouton}}$. Par construction, dans l'ordonnancement $\text{SPT}_{\text{glouton}}$, la date de fin de la $i^{\text{ème}}$ tâche est la date de fin de la tâche i, et la tâche i commence après les tâches dont les numéros d'identification sont compris entre 1 et $i-1$, et avant les tâches dont les numéros sont entre $i+1$ et n. Puisque $\text{SPT}_{\text{glouton}}$ est un algorithme glouton, la plus grande date de fin possible pour la $i^{\text{ème}}$ tâche est atteinte quand les $(i-1)^{\text{ème}}$ premières tâches sont terminées au moment où la $i^{\text{ème}}$ tâche commence à être exécutée : dans ce cas là, la date de fin de la tâche i est $\frac{\sum_{j=1}^{i-1} l_j}{m} + l_i$. La plus grande date de fin possible pour la $i^{\text{ème}}$ tâche dans un ordonnancement $\text{SPT}_{\text{glouton}}$ est donc $\frac{\sum_{j=1}^{i-1} l_j}{m} + l_i$.

Cherchons maintenant la plus petite date de fin que peut prendre la $i^{\text{ème}}$ coordonnée de $\overrightarrow{Y}$, un vecteur de dates de fin d'un ordonnancement de $\mathcal{I}$ ordonné par dates croissantes. Nous remarquons que cette valeur ne peut pas être inférieure à l_i : en effet $\overrightarrow{Y}_i$ est la date de fin de la $i^{\text{ème}}$ tâche et est donc supérieure ou égale à $(i-1)$ autres dates de fin, et l_i est la longueur de la $i^{\text{ème}}$ plus petite tâche. Nous remarquons également que $\overrightarrow{Y}_i$ ne peut pas être inférieure à $\frac{\sum_{j=1}^{i} l_j}{m}$: en effet ceci est la plus petite date de fin des i plus petites tâches (lorsque aucune machine n'est inactive). C'est pourquoi nous avons :

$$\text{RM} \leq \frac{\frac{\sum_{j=1}^{i-1} l_j}{m} + l_i}{\max\left(\frac{\sum_{j=1}^{i} l_j}{m}\ ,\ l_i\right)}.$$

Soit $A = \frac{\sum_{j=1}^{i} l_j}{m}$. Nous considérons les deux cas possibles :

– Supposons que $l_i \geq A$, nous avons :

$$\text{RM} \leq \frac{\frac{\sum_{j=1}^{i-1} l_j}{m} + l_i}{l_i} \leq \frac{A - \frac{l_i}{m} + l_i}{l_i} \leq \frac{A}{l_i} + 1 - \frac{1}{m} \leq 2 - \frac{1}{m}$$

– Supposons que $l_i < A$, nous avons :

$$\text{RM} \leq \frac{\frac{\sum_{j=1}^{i-1} l_j}{m} + l_i}{\frac{\sum_{j=1}^{i} l_j}{m}} \leq \frac{A - \frac{l_i}{m} + l_i}{A} \leq 1 + \frac{l_i}{A}(1 - \frac{1}{m}) \leq 2 - \frac{1}{m}$$

Dans les deux cas, nous avons $\text{RM} \leq 2 - \frac{1}{m}$. Puisque $\text{RM} = c_{gbl}(X_{SPT})$, et puisque X_{SPT} est le vecteur des dates de fin de l'ordonnancement $\text{SPT}_{\text{glouton}}$ de $\mathcal{I}$, nous avons montré qu'un ordonnancement X retourné par $\text{SPT}_{\text{glouton}}$ verifie $c_{gbl}(X) \leq 2 - \frac{1}{m}$, et donc que $c^*_{gbl}(\mathcal{I}) \leq 2 - \frac{1}{m}$. □

Nous montrons maintenant qu'il n'est pas possible d'avoir $c^*_{gbl}(\mathcal{I}) < 2 - \frac{1}{m}$ pour toutes les instances $\mathcal{I}$ de $(P \parallel all)$.

Théorème 4.4 *Il n'est pas possible d'avoir $c^*_{gbl}(\mathcal{I}) < 2 - \frac{1}{m}$ pour toute instance $\mathcal{I}$ (avec $m \geq 2$) de $(P \parallel all)$.*

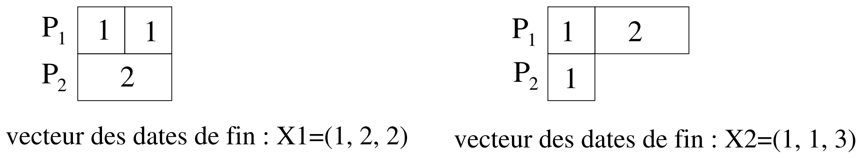


FIG. 4.3 – Exemple dans lequel nous avons $c^*_{gbl}(\mathcal{I}) = 2 - \frac{1}{m}$ pour $m = 2$.

Preuve : Considérons l'instance suivante $\mathcal{I}$: m machines et $m\,(m-1)$ tâches de taille 1 et une tâche de taille m (voir figure 4.3 dans le cas où $m = 2$). Considérons le vecteur des dates de fin X_1 obtenu avec $\mathrm{SPT}_{\mathrm{glouton}}$: les m premières coordonnées sont égales à 1 ; pour tout $i \in \{2, \ldots, m-1\}$, les coordonnées entre $(i\,m+1)$ et $(i+1)\,m$ sont égales à $(i+1)$; et la coordonnée $m\,(m-1)+1$ est égale à $(m-1)+m = 2\,m-1$. Dans n'importe quel ordonnancement dans lequel les m premières tâches ne sont pas des tâches de longueur 1, la tâche de longueur m est l'une de ces m premières tâches et celle-ci a une date de fin égale à $m \geq 2$, et on a donc un rapport d'approximation global supérieur ou égal à 2. De même, pour tout $i \in \{2, \ldots, m-1\}$, dans n'importe quel ordonnancement dans lequel les tâches entre les positions $(i\,m+1)$ et $(i+1)\,m$ ne sont pas toutes des tâches de longueur 1, il y a la tâche de taille m dont la date de fin est supérieure ou égale à $(i+m)$ et le rapport d'approximation global est supérieur ou égal à $(i+m)/(i+1) \geq 2 - 1/m$ car $i \leq m-1$. Notons que le rapport d'approximation global de l'ordonnancement retourné par $\mathrm{SPT}_{\mathrm{glouton}}$ est égal à $2 - 1/m$ car la $n^{\text{ème}}$ tâche se termine à la date $(m-1)+m = 2\,m-1$ alors que la date de fin de la $n^{\text{ème}}$ tâche de l'ordonnancement dans lequel la tâche de longueur m occupe une machine à elle seule, et où il y a m tâches de taille 1 sur les autres machines, se termine à la date m. Le rapport entre les deux est alors de $(2\,m-1)/m = 2 - 1/m$. Ainsi, pour chaque vecteur $Y \in V(\mathcal{I})$, nous avons $c_{gbl}(Y) \geq 2 - \frac{1}{m}$, et donc $c^*_{gbl}(\mathcal{I}) = 2 - \frac{1}{m}$. □

4.3.2 Mesure d'équité individuelle

Dans cette section, nous comparons la date de fin de chaque tâche dans la solution retournée par $\mathrm{SPT}_{\mathrm{glouton}}$ à la meilleure date de fin que cette tâche aurait pu avoir dans un autre ordonnancement.

Théorème 4.5 *Soit $\mathcal{I}$ une instance de n tâches à ordonnancer sur $m \geq 1$ machines. Le rapport de satisfaction individuel du problème $(P \parallel all)$ est égal à $c^*_{ind}(\mathcal{I}) \leq 1 + \frac{n-1}{m}$. De plus, le vecteur des dates de fin X de l'ordonnancement retourné par $SPT_{glouton}$ vérifie $c_{ind}(X) \leq 1 + \frac{n-1}{m}$.*

Preuve : Considérons une instance $\mathcal{I}$ dans laquelle les tâches $1, \ldots, n$ sont ordonnées par longueurs croissantes. Soit X_{SPT} l'ordonnancement que $\mathrm{SPT}_{\mathrm{glouton}}$ retourne pour l'instance $\mathcal{I}$. $c_{ind}(X_{SPT})$ est le rapport maximum entre X_{SPT} et n'importe quel ordonnancement Y de l'instance $\mathcal{I}$. Montrons que $c_{ind}(X_{SPT})$ ne peut pas être supérieur à $1 + \frac{n-1}{m}$. Considérons que ce rapport correspond à la valeur de la $i^{\text{ème}}$ coordonnée de X_{SPT} divisée par la valeur de la $i^{\text{ème}}$ coordonnée de Y ($i \in \{1, \ldots, n\}$).

Le rapport maximal peut être atteint si la date de fin de la tâche i est la plus grande possible dans l'ordonnancement $\mathrm{SPT}_{\mathrm{glouton}}$. Puisque $\mathrm{SPT}_{\mathrm{glouton}}$ est un algorithme glouton, ce cas arrive lorsque les $(i-1)$ premières tâches sont déjà terminées quand la tâche i commence son exécution : la date de fin de la tâche i est alors égale à $\frac{\sum_{j=1}^{i-1} l_j}{m} + l_i$. La valeur maximale que peut prendre la $i^{\text{ème}}$ coordonnée de X_{SPT} est donc $\frac{\sum_{j=1}^{i-1} l_j}{m} + l_i$. La valeur minimale que peut prendre la $i^{\text{ème}}$ coordonnée de Y est l_i : ceci est le cas dans un ordonnancement dans lequel la tâche i est ordonnancée en première position. Nous avons alors :

$$c_{ind}(X_{SPT}) \leq \frac{\frac{\sum_{j=1}^{i-1} l_j}{m} + l_i}{l_i} \leq 1 + \frac{\sum_{j=1}^{i-1} l_j}{m\, l_i} \leq 1 + \frac{(i-1)\, l_i}{m\, l_i} \leq 1 + \frac{n-1}{m}.$$

Puisque $c_{ind}(X_{SPT}) \leq 1 + \frac{n-1}{m}$, nous avons $c^*_{ind}(\mathcal{I}) \leq 1 + \frac{n-1}{m}$. □

Nous montrons maintenant qu'il n'existe pas d'algorithme avec un rapport de satisfaction individuel inférieur à $1 + \frac{n-1}{m}$ quelque soit l'instance considérée.

Théorème 4.6 *Il existe une instance $\mathcal{I}$ de n tâches à ordonnancer sur m machines telle que $c^*_{ind}(\mathcal{I}) = 1 + \frac{n-1}{m}$ pour le problème $(P \parallel all)$.*

Preuve : Considérons l'instance $\mathcal{I}$ suivante : m machines, et $n = m+1$ tâches de longueur 1. Nous avons $c^*_{ind}(\mathcal{I}) \geq 2 = 1 + \frac{n-1}{m}$ car, quel que soit l'ordonnancement de ces tâches, au moins une tâche aura une date de fin supérieure ou égale à 2. Puisque nous avons vu dans le théorème 6.12 que $c^*_{ind}(\mathcal{I}') \leq 1 + \frac{n-1}{m}$ quelque soit l'instance $\mathcal{I}'$ considérée nous en déduisons que $c^*_{ind}(\mathcal{I}) = 1 + \frac{n-1}{m}$. □

4.3.3 Mesure d'équité indviduelle parmi les ordonnancements SPT

Dans cette section, nous comparons la date de fin de chaque tâche dans un ordonnancement SPT X à la meilleure date de fin que cette tâche pourrait avoir dans un autre ordonnancement SPT Y.

Nous utiliserons les notations introduites dans les sections précédentes. De plus, nous définissons $V_{SPT}(\mathcal{I})$ comme étant l'ensemble des vecteurs des dates de fin induits par des ordonnancements SPT de l'instance $\mathcal{I}$. Soit $X \in V_{SPT}(\mathcal{I})$. Nous définissons $c_{SPT}(X)$ comme étant le plus petit α tel que $X \preccurlyeq \alpha Y$ pour tout $Y \in V_{SPT}(\mathcal{I})$. De façon informelle, ceci peut être vu comme le rapport de satisfaction individuel de X par rapport aux autres vecteurs de V_{SPT} : c'est le plus petit α pour lequel X est α-approché de tout vecteur $Y \in V_{SPT}(\mathcal{I})$. Le meilleur rapport possible pour l'instance $\mathcal{I}$ est alors

$$c^*_{SPT}(\mathcal{I}) = \min_{X \in V_{SPT}(\mathcal{I})} c_{SPT}(X).$$

Théorème 4.7 *Soit $\mathcal{I}$ une instance de n tâches à ordonnancer sur $m \geq 1$ machines. Le rapport de satisfaction individuel du problème $(P \parallel all)$ par rapport aux ordonnancements SPT est égal à $c^*_{SPT}(\mathcal{I}) \leq 2$. De plus, le vecteur des dates de fin X de n'importe quel ordonnancement SPT vérifie $c_{SPT}(X) \leq 2$.*

Preuve : Considérons une instance $\mathcal{I}$ dans laquelle les tâches $1, \ldots, n$ sont ordonnées par longueurs croissantes. $c_{SPT}(X)$ est le rapport maximal entre un ordonnancement SPT X et n'importe quel autre ordonnancement SPT Y. Montrons que $c_{SPT}(X)$ ne peut pas être supérieur à 2. Considérons que ce rapport est égal à la $i^{\text{ème}}$ coordonnée de X, divisée par la $i^{\text{ème}}$ coordonnée de Y ($i \in \{1, \ldots, n\}$). Soit P_{iX} (respectivement P_{iY}) la machine sur laquelle la tâche i est ordonnancée dans la solution X (respectivement Y), et soit r_i le rang auquel la tâche i est ordonnancée.

Le rapport le pire est atteint quand la date de fin de la tâche i est aussi grande que possible dans l'ordonnancement X. Ce cas arrive lorsque les tâches avant i sur P_{iX} sont aussi grandes que possible : la plus grande tâche de chaque rang inférieur à r_i est sur P_{iX} dans la solution X. Soit B_m la date de fin de ces tâches (i.e. B_m est égale à la date de début de i dans X). B_m est égale à la date de fin des $(i-1)$ premières tâches de la machine P_m dans un ordonnancement $\text{SPT}_{\text{glouton}}$. De la même façon, le rapport le pire est atteint quand la date de fin de la tâche i est la plus petite possible dans l'ordonnancement Y : la plus petite tâche de chaque rang inférieur à r_i est sur P_{iY} dans la solution Y. Soit B_1 la date de fin de ces tâches (i.e. B_1 est égale à la date de début de i dans Y). B_1 est égale à la date de fin des $(i-1)$ premières tâches de la machine P_1 dans un ordonnancement $\text{SPT}_{\text{glouton}}$. La différence maximale entre la date de fin de la tâche i dans X et dans Y est égale à $\delta = B_m - B_1$. Ce nombre est inférieur ou égal à la longueur de chaque tâche du rang r_i, et est donc inférieur ou égal à l_i, la longueur de la tâche i (cette propriété est une conséquence directe du lemme 4.1). Nous avons donc :

$$c_{SPT}(X) \leq \frac{B_m + l_i}{B_1 + l_i} \leq \frac{B_1 + \delta + l_i}{B_1 + l_i} \leq 2.$$

Quelque soit l'ordonnancement SPT X d'une instance $\mathcal{I}$, nous avons : $c_{SPT}(X) \leq 2$, et donc $c^*_{SPT}(\mathcal{I}) \leq 2$. □

Montrons maintenant que cette borne est la meilleure possible.

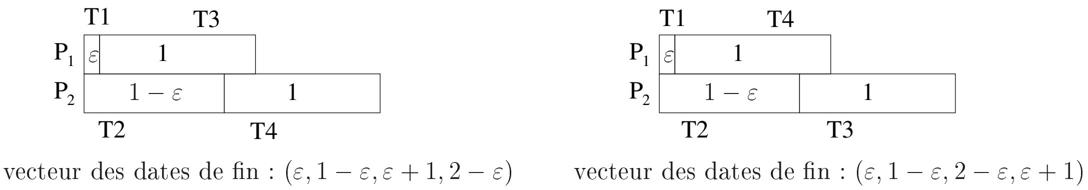


FIG. 4.4 – Exemple dans lequel $c^*_{SPT}(\mathcal{I})$ tend vers 2 quand ε tend vers 0. Le nombre à l'intérieur de chaque tâche représente sa longueur, et l'inscription T_i à côté d'une tâche indique la tâche i.

Théorème 4.8 *Soit $\varepsilon > 0$. Il existe une instance $\mathcal{I}$ de n tâches à ordonnancer sur m machines telle que $c^*_{SPT}(\mathcal{I}) > 2 - \varepsilon$ pour le problème $(P \parallel all)$.*

Preuve : Pour des raisons de lisibilité, nous donnons cette preuve dans le cas où il y a deux machines et nous expliquons à la fin comment la généraliser dans le cas de m machines. Considérons l'instance $\mathcal{I}$ suivante : une tâche 1 de longueur ε, une tâche 2 de longueur $1 - \varepsilon$, et deux tâches 3 et

4 de longueur 1. Nous supposons que $\varepsilon < 1$. Les vecteurs des dates de fin correspondant aux deux ordonnancements SPT possibles sont $X_1 = (\varepsilon, 1-\varepsilon, 1+\varepsilon, 2-\varepsilon)$ et $X_2 = (\varepsilon, 1-\varepsilon, 2-\varepsilon, 1+\varepsilon)$ (voir figure 4.4). Nous avons : $c^*_{SPT}(\mathcal{I}) = c_{SPT}(X_1) = c_{SPT}(X_2) = \frac{2-\varepsilon}{1+\varepsilon} < 2-\varepsilon$, ce qui tend vers 2 quand ε tend vers 0.
Si $m > 2$, la preuve reste la même, en ajoutant $m-2$ tâches de longueur $(1-\varepsilon)$ et $m-2$ tâches de longueur 1 à l'instance décrite dans le cas où $m = 2$. □

4.4 Conclusion

Nous avons étudié plusieurs problèmes qui peuvent être vus comme des problèmes d'équité (vis-à-vis des machines ou des tâches) et qui ont pour but de rendre un ordonnancement plus "juste", i.e. de faire en sorte qu'il n'y ait pas de machine ou de tâche trop désavantagée par rapport aux autres.

Nous avons étudié le problème Min Max SDF, qui cherche à minimiser la somme maximum des dates de fin par machine, et montré que ce problème est NP-difficile. Ce résultat permet de comparer la complexité de ce problème à la complexité d'autres problèmes proches : le problème Min SDF, qui cherche à minimiser la somme totale des dates de fin est polynomial, alors que le problème $(P||C_{max})$, qui cherche à minimiser la somme maximum des longueurs par machine, est NP-difficile. Le problème qui consiste à savoir s'il existe un schéma d'approximation pour le problème Min Max SDF reste ouvert.

Nous avons étudié deux mesures d'équité pour les tâches. Nous avons notamment étudié le rapport d'approximation global pour le problème qui consiste à minimiser la date de fin des tâches. Nous avons montré que ce rapport est égal à $2 - 1/m$ pour l'algorithme $\mathrm{SPT}_{\mathrm{glouton}}$. Sachant que le rapport d'approximation (classique) de cet algorithme est égal à $2-1/m$ [42], il n'était pas possible d'avoir un meilleur rapport. De plus nous avons également montré qu'aucun algorithme ne peut avoir, sur toutes les instances, un rapport d'approximation global inférieur à $2 - 1/m$. Nous avons également montré que cet algorithme possède le meilleur rapport d'approximation possible pour le critère de satisfaction individuelle et que l'ensemble des ordonnancements SPT sont 2-approchés les uns des autres vis-à-vis de ce critère.

Jusqu'à présent nous souhaitions seulement que les tâches soient satisfaites (ou pas trop mécontentes) de l'ordonnancement construit. Dans les chapitres suivants nous allons étudier les cas dans lesquels une tâche a plus de liberté d'action : le cas où elle peut refuser d'aller sur une machine qui ne lui convient pas et choisir une autre machine pour diminuer sa date de fin (voir chapitre 5), et le cas où l'algorithme qui construit l'ordonnancement ne connaît pas dès le départ les longueurs des tâches mais où chaque tâche annonce elle-même sa longueur et peut donc mentir sur cette valeur si cela lui permet d'être ordonnancée plus tôt (voir chapitre 6). Nous avons montré que l'algorithme $\mathrm{SPT}_{\mathrm{glouton}}$ possède de bonnes propriétés vis-à-vis de l'équité. Nous verrons dans le chapitre 6 que cet algorithme possède une autre propriété intéressante puisqu'il est à véracité garantie.

Chapitre 5

Ordonnancement de tâches autonomes et stabilité

Nous considérons dans ce chapitre l'ordonnancement de tâches autonomes sur deux machines (et dans une moindre mesure sur m machines), et nous étudions la relation entre la stabilité d'un ordonnancement, i.e. le fait que les tâches n'aient pas intérêt à changer de machine, et la date de fin de cet ordonnancement. Une partie des travaux de ce chapitre a été publiée dans [12].

5.1 Introduction

Nous considérons le problème suivant : nous avons deux (ou m) machines, ayant chacune une politique pour ordonnancer ses tâches, et n tâches autonomes, qui peuvent choisir sur quelle machine elles vont être exécutées. Le but des tâches est de minimiser leur date de fin. Nous nous plaçons dans le modèle CKN. Il est utile ici de rappeller les deux principaux modèles étudiés pour représenter la date de fin d'une tâche (voir section 1.2.1 pour plus de détails) :

- Dans le modèle KP [55], sur une machine donnée, les tâches sont découpées en petites parties puis ordonnancées alternativement. On considère ainsi que toutes les tâches terminent en même temps. Le but d'une tâche est donc de minimiser la date de fin de la machine sur laquelle se trouve sa tâche.
- Dans le modèle CKN [30], les tâches ne sont pas découpées mais exécutées linéairement, du début à la fin de la tâche. Chaque machine a une politique qui lui donne l'ordre dans lequel elle exécute ses tâches. Connaissant cette politique, le but d'une tâche est de minimiser sa date de fin.

Notre but est d'obtenir - de façon centralisée - un ordonnancement $\mathcal{O}$ qui soit *stable*, c'est à dire tel qu'il n'y ait pas d'agent qui ait intérêt à unilatéralement changer de machine. Une tâche, connaissant les longueurs des autres tâches, la machine sur laquelle celles-ci sont affectées (dans $\mathcal{O}$), et la politique des machines, peut en effet calculer la date de fin qu'elle aura si elle reste sur la machine sur laquelle elle a été affectée, et la date de fin qu'elle aura si elle va au contraire sur l'autre machine. L'ordonnancement que nous souhaitons obtenir doit donc être un *équilibre de Nash*. Nous souhaitons également obtenir un ordonnancement ayant une date de fin la plus petite possible : l'ordonnancement obtenu doit donc être le *meilleur* équilibre de Nash vis-à-vis de la date de fin de l'ordonnancement.

Ceci nous amène à la notion de *prix de la stabilité*, qui évalue l'impact (sur une fonction objectif donnée) de ne chercher que des solution stables. Nous avons présenté cette notion dans la section 1.2.5 : le *prix de la stabilité* est le rapport maximum, sur toutes les instances possibles, entre la valeur de la fonction objectif (la date de fin de l'ordonnancement) dans le meilleur équilibre de Nash, et valeur de la fonction objectif dans la meilleure solution (ordonnancement) possible.

Dans le modèle KP, il existe toujours un équilibre de Nash pur qui soit une solution optimale vis-à-vis de la date de fin de l'ordonnancement, et donc le prix de la stabilité est de 1 dans ce modèle. Ce résultat provient directement du fait suivant : il est toujours possible dans le modèle KP de transformer (*nashifier*) un ordonnancement initial en un équilibre de Nash pur, sans augmenter la date de fin de l'ordonnancement [39]. Il est alors intéressant de se demander si l'on peut obtenir le même résultat dans le modèle CKN.
Dans ce modèle, le prix de la stabilité dépend de la politique des machines. Nous supposons tout d'abord (et dans la majeure partie de ce chapitre) que cette politique est LPT : chaque machine ordonnance les tâches qui lui sont affectées de la plus grande à la plus petite, et si deux tâches ont la même longueur, celle qui a le plus petit numéro d'identification est ordonnancée en premier. En effet, cette politique, en plus d'être simple, mène au meilleur équilibre de Nash connu sachant qu'il n'y a pas de communication entre les machines. Dans [30], les auteurs proposent un mécanisme de coordination (voir section 1.2.4) dans lequel chaque machine a comme politique LPT (plus des petits délais afin d'éviter les équilibres de Nash mixtes) : ce mécanisme de coordination est le meilleur connu, et a un rapport de $4/3-1/3m$, soit $7/6$ quand $m = 2$ (les auteurs conjecturent qu'il n'existe pas de mécanisme de coordination avec un meilleur rapport d'approximation). Dans la section 5.3, nous examinerons aussi brièvement le cas où la politique est SPT et nous comparerons ces résultats avec la politique LPT, et dans la section 5.4 nous examinerons également le cas d'une politique probabiliste. Considérons pour l'instant que la politique des machines est LPT.
Si la politique des machines est LPT, le seul équilibre de Nash pur est l'ordonnancement que l'on obtient en utilisant l'algorithme de liste LPT, qui ordonnance de manière gloutonne les tâches de la plus grande à la plus petite, dès qu'une machine est libre (voir section 1.1.1). Ceci nous montre que le prix de la stabilité dans ce cas là est égal au rapport d'approximation de l'algorithme de liste LPT [42], c'est à dire $4/3 - 1/3m$.

Stabilité approchée
Un moyen naturel de diminuer ce rapport est de relâcher la notion de stabilité. Nous considérons maintenant qu'un ordonnancement est stable si c'est un *équilibre de Nash α-approché* (avec $\alpha \geq 1$), c'est à dire un ordonnancement dans lequel aucune tâche n'a suffisamment intérêt à changer (unilatéralement) de machine. Nous dirons qu'une tâche a *suffisamment intérêt* à quitter la machine sur laquelle elle est affectée, si et seulement si ce changement réduit sa date de fin d'un facteur supérieur à α (i.e. si la date de fin de cette tâche serait plus de α fois plus petite sur l'autre machine, plutôt que sur la machine sur laquelle elle est affectée).
Notons que cette définition est différente de la définition d'un ε-équilibre de Nash, introduite dans [60]. Dans [60], une solution est un ε-équilibre de Nash si tout agent, en changeant unilatéralement de stratégie, augmente son profit d'au plus ε : dans une telle solution, le profit d'un agent changeant unilatéralement de stratégie est inférieur ou égal à son profit courant *plus* ε (et non *fois* ε comme dans notre définition). Les ε-équilibres de Nash ont été introduits dans un contexte assez différent du notre, puisque les auteurs ont voulu montrer dans [60] qu'il est possible, pour tout $\varepsilon > 0$ fixé, et

dans un jeu à deux joueurs (agents), d'avoir des ε-équilibres de Nash avec un support logarithmique (en le nombre de stratégies possibles). Le support est le nombre de stratégies choisies par chaque agent ayant une stratégie mixte. Ce résultat montre qu'il est possible d'avoir des solutions très proches d'un équilibre de Nash, en ayant des stratégies assez simples.

Nous définissons maintenant le *prix de la stabilité α-approchée* comme le rapport maximum entre la valeur de la fonction objectif (ici, la date de fin de l'ordonnancement) dans le meilleur équilibre de Nash α-approché, et la valeur de la fonction objectif dans la solution optimale vis-à-vis de cette fonction objectif (ce maximum est pris parmi toutes les instances) : soit D l'ensemble des instances possibles, $\mathcal{N}_\alpha(I)$ l'ensemble des équilibres de Nash α-approchés de l'instance I, $f(e)$ la date de fin de l'ordonnancement e, et $OPT(I)$ la date de fin d'un ordonnancement optimal de l'instance I. Si $\forall I \in D, \mathcal{N}_\alpha(I) \neq \emptyset$, alors le prix de la stabilité α-approchée est le suivant :

$$\text{Prix de la stabilité } \alpha\text{−approchée} = \max_{I \in D} \min_{e \in \mathcal{N}_\alpha(I)} \frac{f(e)}{OPT(I)}$$

Remarquons que si $\alpha = 1$, alors nous obtenons le prix de la stabilité défini précédemment. Quand α tend vers l'infini alors le prix de la stabilité α-approchée tend vers 1 (l'ensemble des équilibres de Nash α-approché est alors égal à l'ensemble des solutions possibles). Si le prix de la stabilité α-approchée est égal à γ, alors cela signifie qu'il n'existe pas d'algorithme (même exponentiel) qui a un rapport d'approximation inférieur à γ et qui retourne toujours des ordonnancements qui sont des équilibres de Nash α-approchés.

Exemple. Supposons que l'on ait deux machines ayant comme politiques LPT, et les tâches suivantes : deux tâches de longueur 3, et trois tâches de longueur 2. Le seul équilibre de Nash pur est l'ordonnancement dans lequel les tâches de longueur 3 sont ordonnancées en premier (au temps 0), et sont ensuite suivies par les tâches de taille 2. Cette solution a une date de fin de 7, alors que la date de fin dans une solution optimale est 6. Soit $\mathcal{O}$ une solution optimale. Dans cette solution, les deux tâches de taille 3 sont obligatoirement sur la même machine, et une tâche de taille 3 commence au temps 3. En changeant de machine, cette tâche deviendrait la plus grande tâche sur sa machine et serait alors ordonnancée au temps 0 : sa date de fin passerait donc de 6 à 3. Comme cette tâche peut réduire sa date de fin d'un facteur de 2 en changeant de machine, on dit que cette tâche est *2-Nash-approchée*. Puisque chaque tâche de $\mathcal{O}$ peut réduire sa date de fin d'un facteur au plus 2 en changeant de machine, cet ordonnancement est un équilibre de Nash 2-approché.

Plan de ce chapitre : Pour des raisons de cohérence et de lisibilité, nous considérons dans ce chapitre qu'il y a deux machines ($m = 2$). Certains résultats sont généralisabes dans le cas où il y a $m > 2$ machines : nous indiquons alors comment généraliser ces résultats après les avoir présentés dans le cas de deux machines.

Le but de ce chapitre est d'étudier la *relation entre la stabilité d'un ordonnancement* (le plus petit nombre α pour lequel cet ordonnancement est un équilibre de Nash α-approché) *et sa date de fin*. Nous étudions tout d'abord la relation entre α et le rapport d'approximation (vis-à-vis de la date de fin de l'ordonnancement), étant donné que la politique des machines est LPT. Dans la section 5.2.1, nous montrons que le prix de la stabilité α-approchée est d'au moins $\frac{8}{7}$ pour tout $\alpha < 2.1$, et d'au plus $\frac{8}{7}$ pour tout $\alpha \geq 3$. Nous donnons un algorithme qui montre cette dernière borne : il a un rapport d'approximation de $\frac{8}{7}$ et retourne des équilibres de Nash 3-approchés. Dans la section 5.2.2 nous montrons que le prix de la stabilité α-approché est inférieur ou égal à $1+\varepsilon$ pour tout $\alpha \geq \frac{1}{\varepsilon}$. Cette borne est obtenue en analysant le schéma d'approximation de Graham légèrement modifié [42]. Dans la section 5.2.3 nous montrons que le prix de la stabilité α-approchée est supérieur ou égal à $1+\varepsilon$ pour tout $\alpha \leq k$, où k est une certaine constante en $\Theta(\varepsilon^{-1/2})$. Nous montrons aussi un résumé des résultats négatifs et positifs obtenus jusqu'alors dans ce chapitre. Nous comparons ces résultats dans la section 5.3 avec ce que l'on obtient si la politique des liens est SPT (i.e. si chaque machine ordonnance ses tâches par ordre de longueurs croissantes). Dans la section 5.4 nous examinons le cas où la politique des liens n'est plus nécessairement déterministe mais peut être probabiliste : le but de chaque tâche est alors de minimiser son espérance de date de fin. Nous concluons ce chapitre dans la section 5.5.

5.2 Etude de la politique LPT

5.2.1 LPT_{swap} : une variante de LPT

Notations : Dans cette section, nous supposons qu'il y a deux machines P_1 et P_2 : nous appellerons P_1 la machine la plus chargée, et P_2 l'autre machine (i.e. la somme des longueurs des tâches de P_1 est supérieure ou égale à la somme des longueurs des tâches de P_2). Soit n_i le nombre de tâches ordonnancées sur P_i. Soit x_i la $i^{\text{ème}}$ tâche de P_1, et y_i la $i^{\text{ème}}$ tâche de P_2. La longueur de la tâche i sera notée l_i.

Soit t_{max} la tâche ordonnancée en dernière position sur P_1 (la machine la plus chargée). On dit qu'une tâche est *grande* si sa longueur est supérieure ou égale à la longueur de t_{max}. Une tâche est *petite* si sa longueur est inférieure à celle de t_{max}.

Nous représentons un ordonnancement par deux ensembles A et B, où A (respectivement B) est l'ensemble des tâches ordonnancée sur P_1 (respectivement sur P_2). Sur chaque machine, les tâches sont ordonnancées de la plus grande à la plus petite. Soit $\xi = (A, B)$ un ordonnancement des n tâches sur les deux machines. Soit a (respectivement b) un sous-ensemble de tâches ordonnancées sur P_1 (respectivement sur P_2). On appelle $swap(\xi, a \leftrightarrow b)$ l'ordonnancement $((A \setminus a) \cup b, (B \setminus b) \cup a)$. Ainsi, dans $swap(\xi, a \leftrightarrow b)$, toutes les tâches sont sur la même machine que dans ξ, sauf les tâches de a et de b qui ont échangé leurs machines. Dans ce nouvel ordonnancement, chaque machine exécute ses tâches en utilisant la politique LPT (si deux tâches ont des longueurs égales, on ordonnance en premier celle qui a le plus petit numéro d'identification).

Considérons maintenant l'algorithme LPT_{swap} donné dans la page suivante.

Théorème 5.1 *L'algorithme LPT_{swap} est $\frac{8}{7}$-approché pour la date de fin de l'ordonnancement.*

Preuve : Supposons que l'on a une instance I de tâches à ordonnancer, et soit ξ l'ordonnancement LPT de ces tâches. Si ξ est $\frac{8}{7}$-approché, alors l'ordonnancement retourné par LPT_{swap} pour l'instance I sera $\frac{8}{7}$-approché, car l'ordonancement retourné par LPT_{swap} est le meilleur parmi plusieurs ordonnancements, dont ξ. Nous regardons maintenant le cas où ξ est un ordonnancement LPT qui n'est pas $\frac{8}{7}$-approché. Nous allons alors montrer que l'algorithme LPT_{swap} transforme cet ordonnancement en un ordonnancement $\frac{8}{7}$-approché.

Ordonnancer les tâches avec l'algorithme de liste LPT. Soit LPT l'ordonnancement ainsi obtenu. Soit $S = \sum_{i=1}^{n} l_i$.

si $n_2 \geq 2$ *et* $((n_1 = 3$ *et* $l_{x_1} + l_{x_2} + l_{x_3} > (\frac{4}{7})S)$ *ou* $(n_1 = 4))$ **alors**

  si $n_1 = 3$ **alors**

    Soit $\xi_1 = swap(LPT, \{x_1\} \leftrightarrow \{y_2\})$, $\xi_2 = swap(LPT, \{x_2\} \leftrightarrow \{y_2\})$ et $\xi_3 = swap(LPT, \{x_1\} \leftrightarrow \{y_1\})$.

    Retourner un ordonnancement qui a la plus petite date de fin parmi les ordonnancements : LPT, ξ_1, ξ_2 et ξ_3.

  fin

  si $n_1 = 4$ **alors**

    Soit $\xi_4 = swap(LPT, \{x_3, x_4\} \leftrightarrow \{y_2\})$.

    Retourner un ordonnancement qui a la plus petite date de fin parmi les ordonnancements : LPT et ξ_4.

  fin

fin

sinon

  Retourner LPT.

fin

L'algorithme LPT_{swap}

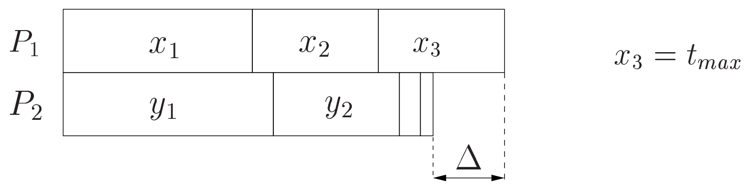


FIG. 5.1 – Un ordonnancement LPT.

Remarque : il y a 3 ou 4 tâches sur P_1 et exactement 2 grandes tâches sur P_2.

Montrons tout d'abord qu'il y a exactement deux grandes tâches sur P_2. Soit Δ la différence entre la date de fin de la dernière tâche de P_1 et celle de la dernière tâche de P_2 (voir Figure 5.1). La date de fin de l'ordonnancement ξ est $\frac{\sum_{i=1}^{n} l_i + \Delta}{2}$. Puisque ξ n'est pas $\frac{8}{7}$-approché, on sait que $\frac{\sum_{i=1}^{n} l_i + \Delta}{2} > \frac{8}{7} OPT$, où OPT est la date de fin d'un ordonnancement optimal.

Puisque OPT est supérieur ou égal à $\frac{\sum_{i=1}^{n} l_i}{2}$, on a $\frac{\Delta}{2} > (\frac{8}{7} - 1)OPT$, et donc $\Delta > \frac{2}{7}\, OPT$. Soit $\epsilon > 0$ tel que $\Delta = (\frac{2}{7} + \epsilon)OPT$.
Nous savons que $l_{t_{max}} \geq \Delta$ (car dans un ordonnancement LPT, à chaque étape, on ajoute une tâche sur la machine la moins chargée), donc chaque grande tâche a une longueur supérieure ou égale à Δ. Nous savons aussi que la date de fin de la dernière tâche sur P_2 est inférieure ou égale à $OPT - (\frac{1}{7} + \epsilon_1)OPT$, avec $\epsilon_1 > 0$, car sinon l'ordonnancement ξ serait $\frac{8}{7}$-approché. Donc le nombre maximum de grandes tâches sur P_2 est égal à $\lfloor \frac{OPT - (\frac{1}{7} + \epsilon_1)OPT}{\Delta} \rfloor \leq \lfloor \frac{1 - (\frac{1}{7} + \epsilon_1)}{\frac{2}{7} + \epsilon} \rfloor = 2$.

De plus, il y a au moins deux grandes tâches sur P_2, car sinon l'ordonnancement ξ serait un ordonnancement optimal. En effet, s'il n'y a pas de grande tâche sur P_2, alors cela signifie qu'il y a seulement une grande tâche dans l'ordonnancement (la seule grande tâche de P_1), et ξ est un ordonnancement optimal. S'il y a seulement une grande tâche t_l sur P_2, alors ξ est aussi optimal, car un ordonnancement dans lequel on aurait t_l avec une grande tâche de P_1 aurait une date de fin au moins aussi grande que celle de ξ (car t_{max} commence au plus tard à la fin de t_l et t_{max} est la plus petite des tâches de P_1).

Montrons maintenant qu'il y a soit 3, soit 4 tâches sur P_1. Il y a au moins 3 tâches sur P_1 car sinon l'ordonnancement ξ serait optimal. En effet, s'il y a seulement une tâche sur P_1, il est trivial de voir que ξ est optimal. S'il y a deux tâches sur P_1, et si $l_{x_1} \leq l_{y_1}$ alors on ne diminuerait pas la date de fin de l'ordonnancement en mettant x_2 et y_1 sur la même machine (car P_1 est la machine la plus chargée).

S'il y a deux tâches sur P_1 et si $l_{x_1} > l_{y_1}$, alors $l_{y_2} \geq l_{x_2}$ (on a un ordonnancement LPT) : si on ajoute x_2 aux grandes tâches de P_2 (y_1 et y_2) on ne diminue pas la date de fin de l'ordonnancement, car x_2 commence avant la date de fin de y_2. Si l'on ajoute à x_1 une des grandes tâches de P_2 (i.e. on échange x_2 avec y_1 ou y_2), on ne diminue pas la date de fin de l'ordonnancement car la longueur de chacune de ces tâches (y_1 ou y_2) est supérieure ou égale à la longueur de x_2.

Montrons maintenant qu'il y a au plus 4 tâches sur P_1.
On sait que $l_{t_{max}} \geq \Delta > \frac{2}{7}OPT$, et chaque tâche de P_1 est supérieure ou égale à t_{max}. Puisque la date de fin d'un ordonnancement LPT sur deux machines est au plus $\frac{7}{6}OPT$ [42] alors le nombre maximum de tâches sur P_1 est $\lfloor \frac{(\frac{7}{6})OPT}{\Delta} \rfloor \leq \lfloor \frac{(\frac{7}{6})OPT}{(\frac{2}{7}+\epsilon)OPT} \rfloor \leq \lfloor \frac{49}{12} \rfloor = 4$.

Nous avons montré que si un ordonnancement LPT n'est pas $\frac{8}{7}$-approché, alors il a 3 ou 4 tâches sur P_1, et au moins deux tâches sur P_2. De plus, $\sum_{i \in P_1} l_i > (\frac{8}{7})OPT \geq \frac{8}{7}(\frac{\sum_{i=1}^{n} l_i}{2}) = \frac{4}{7}\sum_{i=1}^{n} l_i$. Donc toutes les conditions pour rentrer dans la première boucle “si” de l'algorithme LPT_{swap} sont remplies. Nous allons maintenant regarder le cas où il y a 3 tâches sur P_1.

Cas où il y a 3 tâches sur P_1 :

Il y a deux grandes tâches sur P_2, et ces grandes tâches peuvent être suivies par des petites tâches. Calculons maintenant la somme des longueurs de ces petites tâches. Comme il y a 3 tâches sur P_1, la valeur maximum de $l_{t_{max}}$ est $\frac{(\frac{7}{6})OPT}{3} = (\frac{7}{18})OPT$. Les petites tâches commencent après (ou en même temps que) le début de t_{max} et finissent à la fin de t_{max} moins Δ. Donc la somme maximale des longueurs des petites tâches est : $(\frac{7}{18})OPT - \Delta \leq (\frac{7}{18} - \frac{2}{7})OPT < (\frac{1}{7})OPT$.

Nous considérons maintenant seulement les 5 grandes tâches : x_1, x_2, x_3, y_1 et y_2. Soit OPT_5 un ordonnancement optimal de ces tâches. Dans cet ordonnancement il y a 3 tâches sur une machine (sans perte de généralité, sur P_1) et 2 tâches sur l'autre machine (car s'il y avait dans OPT_5 4 tâches sur une machine, et une tâche sur l'autre machine, notre ordonnancement LPT ξ serait une solution optimale). Il n'est pas non plus possible que, dans OPT_5, y_1 et y_2 soient sur la même machine, car sinon la date de fin de OPT_5 ne serait pas plus petite que celle de ξ. De plus il n'est pas possible que x_1, x_2 et y_1 (respectivement x_1, x_2 et y_2) soient sur la même machine car $y_1 \geq x_3$ (respectivement $y_2 \geq x_3$) et $l_{x_1} + l_{x_2} + l_{x_3} \geq (\frac{8}{7})OPT \geq (\frac{8}{7})OPT_5$. De même, il n'est pas non plus possible que x_1, y_1 et x_3 soient sur la même machine, pour la même raison ($y_1 \geq x_2$). Ainsi un échange dans ξ entre x_1 et y_2, ou x_2 et y_2, ou x_1 et y_1, donne OPT_5 (on considère uniquement les grands tâches). Soit ξ_s l'ordonnancement retourné par LPT_{swap} : c'est celui qui minimise la date de fin parmi les ordonnancements issus de ces 3 échanges. Sa date de fin est égale à $OPT_5 \leq OPT$, plus la longueur d'une part des petites tâches. Nous avons montré que la somme totale des longueurs des petites tâches est inférieure à $(\frac{1}{7})OPT$, donc la date de fin de ξ_s est inférieure à $(\frac{8}{7})OPT$. Considérons maintenant le cas où il y a 4 tâches sur P_1 dans ξ.

Cas où il y a 4 tâches sur P_1 :

Nous avons toujours 2 grandes tâches sur P_2, et ces grandes tâches peuvent être suivies par des petites tâches. Calculons la somme des longueurs de ces petites tâches. Il y a 4 tâches sur P_1 donc la valeur maximale de $l_{t_{max}}$ est $\frac{(\frac{7}{6})OPT}{4} = (\frac{7}{24})OPT$. Les petites tâches commencent après ou en même temps que le début de t_{max} et sont terminées avant (ou en même temps que) la fin de t_{max}, moins Δ. Donc la somme maximale des longueurs de ces petites tâches est : $(\frac{7}{24})OPT - \Delta \leq (\frac{7}{24} - \frac{2}{7})OPT < (\frac{1}{7})OPT$.

Nous allons maintenant considérer uniquement les 6 grandes tâches : x_1, x_2, x_3, x_4, y_1 et y_2. Soit OPT_6 un ordonnancement optimal de ces tâches. Il n'est pas possible qu'il y ait 4 tâches ou plus sur la même machine dans OPT_6, car la longueur de chaque grande tâche est supérieure ou égale à la longueur de t_{max}, qui est supérieure ou égale à Δ, qui est supérieur à $(\frac{2}{7})OPT$, et la somme des longueurs des tâches d'une machine dans OPT_6 est inférieure ou égale à OPT. Il y a donc 3 tâches sur chaque machine dans OPT_6.

Il n'est pas possible que, dans OPT_6, y_1 et y_2 soient sur la même machine, car dans ce cas la date de fin de OPT_6 ne serait pas plus petite que celle de ξ. Il y a donc au plus 6 ordonnancements possibles pour OPT_6 : $s_1 = (\{y_1, x_1, x_2\}, \{y_2, x_3, x_4\})$, $s_2 = (\{y_1, x_1, x_3\}, \{y_2, x_2, x_4\})$, $s_3 = (\{y_1, x_1, x_4\}, \{y_2, x_2, x_3\})$, $s_4 = (\{y_1, x_2, x_3\}, \{y_2, x_1, x_4\})$, $s_5 = (\{y_1, x_2, x_4\}, \{y_2, x_1, x_3\})$, et $s_6 = (\{y_1, x_3, x_4\}, \{y_2, x_1, x_2\})$. Nous allons maintenant montrer que le seul ordonnancement possible pour OPT_6 est s_6.

Soit C_{max} la date de fin de ξ. Nous savons, par construction, que $l_{y_1} + l_{y_2} + l_{x_4} \geq C_{max} > OPT$. Puisque $l_{x_1} \geq l_{y_2}$ (ordonnancement LPT), alors $l_{y_1} + l_{x_1} + l_{x_4} > OPT \geq OPT_6$, et alors l'ordonnancement s_3 ne peut pas être un ordonnancement optimal pour les 6 grandes tâches. Comme $l_{x_2} \geq l_{x_4}$ (respectivement $l_{x_3} \geq l_{x_4}$), nous avons : $l_{y_1} + l_{x_1} + l_{x_2} > OPT \geq OPT_6$ (respectivement $l_{y_1} + l_{x_1} + l_{x_3} > OPT \geq OPT_6$), et donc l'ordonnancement s_1 (respectivement s_2) ne peut pas être un ordonnancement optimal.
Pour montrer que s_4 et s_5 ne sont pas non plus des ordonnancements optimaux, considérons ces deux cas :

- *Cas où* $l_{y_2} > l_{x_2}$. Puisque $l_{y_2} > l_{x_2}$, nous savons que $l_{x_1} \geq l_{y_1}$ (ordonnancement LPT). Nous avons : $l_{y_2} + l_{x_1} + l_{x_4} \geq l_{y_2} + l_{y_1} + l_{x_4}$, qui est supérieur à OPT. Donc s_4 n'est pas un ordonnancement optimal. Comme $l_{x_3} \geq l_{x_4}$, $l_{y_2} + l_{x_1} + l_{x_3} \geq l_{y_2} + l_{x_1} + l_{x_4} > OPT$: s_5 n'est pas un ordonnancement optimal.
- *Cas où* $l_{y_2} \leq l_{x_2}$. Comme $l_{x_2} \geq l_{y_2}$, nous savons que $l_{y_1} + l_{x_2} + l_{x_4} \geq l_{y_1} + l_{y_2} + l_{x_4} > OPT$, et donc s_5 n'est pas un ordonnancement optimal. Comme $l_{x_3} \geq l_{x_4}$, nous savons que $l_{y_1} + l_{x_2} + l_{x_3} \geq l_{y_1} + l_{x_2} + l_{x_4} > OPT$, et donc s_4 n'est pas un ordonnancement optimal.

Donc s_6 est le seul ordonnancement optimal pour les 6 grandes tâches de ξ. Ainsi un échange dans ξ entre y_2 et $\{x_3, x_4\}$ donne l'ordonnancement optimal pour les grandes tâches de ξ. Comme la somme des longueurs des petites tâches de ξ est inférieure à $(\frac{1}{7})OPT$, la date de fin de l'ordonnancement retourné par LPT_{swap} est inférieure à $(\frac{8}{7})OPT$. $\square$

$\xi_1 = (\{\mathbf{y_2}, x_2, x_3\}, \{\mathbf{x_1}, y_1, T\})$
$\xi_2 = (\{x_1, \mathbf{y_2}, x_3\}, \{y_1, \mathbf{x_2}, T\})$
$\xi_3 = (\{\mathbf{y_1}, x_2, x_3\}, \{x_1, \mathbf{y_2}, T\})$
$\xi_4 = (\{x_1, x_2, \mathbf{y_2}\}, \{y_1, \mathbf{x_3}, \mathbf{x_4}, T\})$

TAB. 5.1 – Ordonnancements ξ_1, ξ_2, ξ_3 et ξ_4. T représente l'ensemble des petites tâches. Les tâches en gras sont celles qui ont été changées de machine par l'algorithme LPT_{swap}.

Théorème 5.2 *L'ordonnancement retourné par LPT_{swap}, sur deux machines dont les politiques sont LPT, est un équilibre de Nash 3-approché.*

Preuve : Supposons que l'on ait n tâches à ordonnancer. Soit ξ l'ordonnancement LPT de ces tâches et ξ_s l'ordonnancement retourné par LPT_{swap}. On note f_ξ la date de fin de l'ordonnancement ξ, et f_{ξ_s} la date de fin de ξ_s. Si $\xi_s = \xi$, alors ξ_s est un ordonnancement LPT et est un équilibre de Nash. Sinon $\xi_s \neq \xi$, et l'algorithme LPT_{swap} a effectué un échange : ξ_s est donc égal à ξ_1, ξ_2, ξ_3, ou bien ξ_4. Nous allons montrer que chaque tâche de ξ_s ne diminuera pas sa date de fin d'un facteur de 3 en changeant de machine. Dans la suite de cette preuve nous dirons qu'une tâche n'a pas intérêt à changer de machine si et seulement si elle ne terminera pas plus de 3 fois plus tôt en changeant de machine.

Tout d'abord, les petites tâches (les tâches qui sont plus petites que t_{max}) n'ont pas intérêt à changer de machine, car, si elles changent de machine, elles commenceront dans tous les cas après au moins trois grandes tâches, c'est à dire après un temps supérieur à $\frac{f_{\xi_s}}{3}$, et si elles ne changent pas de machine elles seront terminées avant ou en f_{ξ_s}. De même, il est trivial que les tâches qui sont ordonnancées au tout début de l'ordonnancement n'ont pas intérêt à changer de machine.

Montrons maintenant, en analysant les différents cas possibles, que toutes les grandes tâches de ξ_1, ξ_2, ξ_3 ou ξ_4 n'ont pas intérêt à changer de machine. Le tableau 5.1 rappelle la forme de ces ordonnancements.

Nous considérons tout d'abord les ordonnancements ξ_1, ξ_2 et ξ_3 (i.e. LPT_{swap} a fait un échange entre une tâche de P_1 et une tâche de P_2).

Si $\xi_s = \xi_3$ alors x_1 et y_1 sont ordonnancées en première position et donc il n'ont pas intérêt à changer de machine. Si x_2 ou x_3 (respectivement y_2) change unilatérallement de machine, alors elle sera ordonnancée après x_1 (respectivement y_1). Comme $l_{x_1} \geq \frac{1}{3} f_\xi$ (car x_1 est la plus grande des trois tâches de P_1 dans ξ), si x_2 ou x_3 change de machine, elle va commencer (et être terminée) après $\frac{1}{3} f_\xi \geq \frac{1}{3} f_{\xi_s}$. Comme x_2 et x_3 sont terminées avant f_{ξ_s} dans ξ_s, elles ne vont pas améliorer leur date de fin d'un facteur supérieur à 3 en changeant de machine. De même, comme $l_{y_1} \geq \frac{1}{3} f_\xi$ (car $l_{y_1} + l_{y_2} \geq l_{x_1} + l_{x_2} \geq \frac{2}{3} f_\xi$ et $l_{y_1} \geq l_{y_2}$), y_2 n'a pas intérêt à changer de machine.

Montrons maintenant que les tâches de ξ_1 et ξ_2 n'ont pas intérêt à changer de machine. Soit t une tâche que LPT_{swap} a changé de machine. Si t est sur P_1 (respectivement P_2) dans ξ_s et veut aller sur P_2 (respectivement P_1) elle ne va pas être terminée avant la date à laquelle elle était terminée dans ξ (i.e. avant l'échange de tâche), car les tâches qui étaient avant elle dans ξ sont toujours sur P_2 (respectivement P_1) dans ξ_s. Si t se termine après la date $(\frac{1}{3}) f_\xi$ dans ξ, alors elle

n'a pas intérêt à changer de machine dans ξ_s car t ne sera pas terminée après f_{ξ_s} dans ξ_s et $f_{\xi_s} \leq f_\xi$. Montrons maintenant que les tâches qui ont été échangées dans ξ_1 et ξ_2 : x_1, x_2 et y_2, terminaient après $(\frac{1}{3})f_\xi$ dans ξ.
Si ξ_s est égal à ξ_1 ou ξ_2, alors il y avait 3 tâches sur P_1 dans ξ, et la somme des longueurs de ces tâches est supérieure à $\frac{4}{7}\sum_{i=1}^n l_i$ (ces conditions doivent être respectées pour que LPT_{swap} fasse l'échange). Ainsi $l_{x_3} > \frac{1}{7}\sum_{i=1}^n l_i$, car $x_3 = t_{max}$ et $l_{t_{max}} \geq \Delta > \frac{1}{7}\sum_{i=1}^n l_i$. Comme y_1 est sur P_2, nous savons que $f_{\xi_s} \leq (\sum_{i=1}^n l_i) - l_{y_1}$. Comme $l_{y_1} \geq l_{x_3} > \frac{1}{7}\sum_{i=1}^n l_i$, nous avons $f_{\xi_s} < \frac{6}{7}\sum_{i=1}^n l_i$. Puisque les tâches sont ordonnancées par ordre décroissant, $l_{x_1} \geq \frac{f_\xi}{3} \geq \frac{f_{\xi_s}}{3}$. Donc x_1 est terminée après $\frac{f_{\xi_s}}{3}$ dans ξ. Comme x_2 est terminée après x_1 dans ξ, elle est aussi terminée après $\frac{f_{\xi_s}}{3}$ dans ξ. Nous allons maintenant montrer que y_2 est aussi terminée après $\frac{f_{\xi_s}}{3}$ dans ξ. Si $l_{x_1} \leq l_{y_1}$ alors y_2 est terminée après la fin de x_1 et donc après $\frac{f_{\xi_s}}{3}$. Montrons maintenant qu'il n'est pas possible que $l_{x_1} > l_{y_1}$ et y_2 finisse avant $\frac{f_{\xi_s}}{3}$. Supposons que ceci soit vrai. Puisque y_2 est terminée avant la fin de x_1, $l_{y_2} \geq l_{x_3}$. Comme y_2 est terminée avant $\frac{f_{\xi_s}}{3}$ et après y_1 (et $l_{y_1} \geq l_{y_2}$), $l_{y_2} \leq \frac{f_{\xi_s}}{6}$. Donc $l_{x_3} \leq \frac{f_{\xi_s}}{6} \leq \frac{1}{6}(\frac{6}{7}\sum_{i=1}^n l_i) \leq (\frac{1}{7})\sum_{i=1}^n l_i$. Or nous avons déjà montré que $l_{x_3} > \frac{1}{7}\sum_{i=1}^n l_i$. Donc y_2 est terminée après $\frac{f_{\xi_S}}{3}$ dans ξ.

Nous allons maintenant montrer que les tâches qui n'ont pas été échangées dans ξ_1 et dans ξ_2 n'ont pas intérêt à changer de machine. La tâche x_3 n'a pas intérêt à changer de machine (en changeant elle se retrouverait derrière deux tâches plus grandes qu'elle). De même, x_1 et y_1, qui sont au début de l'ordonnancement dans ξ_2 n'ont pas intérêt à changer de machine. Il reste donc à montrer que y_1 et x_2 n'ont pas intérêt à changer de machine dans ξ_1. Si elle change de machine, x_2 va commencer après y_1 et x_1 et donc après $\frac{f_{\xi_s}}{3}$. Donc elle ne diminuera pas sa date de fin d'un facteur supérieur à 3 en changeant de machine. Nous savons que l_{y_1} est supérieur ou égal à $\frac{f_{\xi_s}}{3}$ (car $l_{y_1} + l_{y_2} \geq f_{\xi_s} - l_{x_3} \geq \frac{2}{3}f_{\xi_s}$ et $l_{y_1} \geq l_{y_2}$) et que y_1 est terminée avant f_{ξ_s} dans ξ_s. Si elle change de machine, elle ne sera pas terminée avant $\frac{f_{\xi_s}}{3}$ et donc n'améliorera pas sa date de fin d'un facteur supérieur à 3.

Considérons maintenant l'ordonnancement $\xi_s = \xi_4$. Cet ordonnancement est obtenu à partir de ξ, dans lequel il y a 4 tâches sur P_1, en échangeant x_3 et x_4 avec y_2. Si x_3 ou x_4 retourne sur P_1 après l'échange, elle sera ordonnancée après x_1 et x_2, qui ne sont pas terminées avant $\frac{f_\xi}{2} > \frac{f_{\xi_s}}{3}$. Montrons que y_2 termine aussi après $\frac{f_{\xi_s}}{3}$ dans ξ. En effet, dans ξ, y_2 est terminée au plus tôt au début de x_4. Puisque $l_{x_4} \leq \frac{f_\xi}{4}$ (ordonnancement LPT), la date de fin de y_2 est supérieure ou égale à $\frac{3f_\xi}{4}$, qui est plus grand que $\frac{f_{\xi_s}}{3}$. Montrons que les autres tâches de ξ_s n'ont pas intérêt à changer de machine. Il est trivial que x_1 et y_1 n'ont pas intérêt à changer de machine car elles sont toutes les deux en première position sur leurs machines. x_2 n'a pas intérêt à changer de machine car, dans ξ_s elle ne sera pas terminée après f_{ξ_s}. Si elle change de machine, elle sera ordonnancée après y_1. Nous savons que y_2 est terminée dans ξ au plus tôt en $\frac{3f_\xi}{4} \geq \frac{3f_{\xi_s}}{4}$, et $l_{y_2} \leq l_{y_1}$. Donc $l_{y_1} \geq \frac{1}{2}(\frac{3f_{\xi_s}}{4}) > \frac{f_{\xi_s}}{3}$. Nous avons montré que toutes les grandes tâches n'ont pas intérêt à changer de machine, et comme ceci est vrai aussi pour les petites tâches, toutes les tâches n'ont pas intérêt à changer de machine dans ξ_s. □

Nous pouvons déduire des deux théorèmes précédents le résultat suivant :

Corollaire 5.1 *Dans le cas de deux machines ayant comme politique LPT, le prix de la stabilité α-approchée est d'au plus $\frac{8}{7}$, pour tout $\alpha \geq 3$.*

Nous avons montré que le prix de la stabilité α-approchée est d'au plus $\frac{8}{7}$ pour tout $\alpha \geq 3$. Montrons maintenant qu'il est d'au moins $\frac{8}{7}$ pour tout $\alpha < 2.1$, ce qui signifie qu'il n'existe pas d'algorithme $\frac{8}{7}$-approché qui retourne des équilibres de Nash α-approchés, avec $\alpha < 2.1$.

Théorème 5.3 *Soit $\varepsilon > 0$. Dans le cas de deux machines ayant comme politique LPT, le prix de la stabilité α-approchée est d'au moins $\frac{8}{7}$, pour tout $\alpha \leq 2.1 - \varepsilon$.*

Preuve : Soit $\varepsilon > 0$. Considérons les tâches suivantes : une tâche de longueur $3.3 - \varepsilon$, une tâche de longueur $3 + \varepsilon$, et trois tâches de longueur 2.1. Dans un ordonnancement optimal (pour la date de fin) de ces tâches, les tâches de longueur 2.1 sont nécessairement sur la même machine, et les deux autres tâches sont sur l'autre machine (voir Figure 5.2 *Gauche*). La date de fin de cet ordonnancement est $OPT = 6.3$. Dans cet ordonnancement, la date de fin de la tâche de longueur $3+\varepsilon$ est 6.3 mais cette tâche pourrait être ordonnancée en première position si elle allait sur l'autre machine (car les politiques des machines sont LPT), et sa date de fin serait alors $3 + \varepsilon$. Donc cet ordonnancement est $\frac{6.3}{3+\varepsilon} > (2.1 - \varepsilon)$-Nash approché.

FIG. 5.2 – *Gauche :* Ordonnancement optimal vis-à-vis de la date de fin. C'est un équilibre de Nash $\frac{6.3}{3+\varepsilon}$-approché. *Droite :* Ordonnancement approché pour la date de fin.

Tous les autres ordonnancements ont une date de fin supérieure ou égale à $7.2 + \varepsilon > \frac{8}{7}OPT$ (voir Figure 5.2 *Droite*). Ainsi, si nous voulons un ordonnancement $\frac{8}{7}$-approché pour la date de fin, alors cet ordonnancement ne sera pas un équilibre de Nash α-approché, avec $\alpha < 2.1 - \epsilon$. Le prix de la stabilité α-approchée est donc supérieur ou égal à $\frac{8}{7}$, pour tout $\alpha \leq 2.1 - \varepsilon$. □

Nous avons vu jusqu'à présent que le prix de la stabilité α-approché est inférieur ou égal à $\frac{8}{7}$ pour tout $\alpha \geq 3$ (i.e. si nous souhaitons un rapport d'approximation de $\frac{8}{7}$, alors nous pouvons avoir un algorithme qui retourne des équilibres de Nash 3-approchés), et supérieur à $\frac{8}{7}$ pour tout $\alpha < 2.1$. Il peut maintenant être intéressant d'étudier la relation entre le rapport d'approximation et α, i.e. ce qu'il se passe quand on considère d'autres valeurs pour le rapport d'approximation que l'on souhaite avoir, ou d'autres valeurs de α. En particulier, est-ce qu'il existe des algorithmes avec un rapport d'approximation aussi petit que possible et qui retourne des équilibres de Nash α-approchés, avec α borné (et aussi petit que possible) ? La section suivante donne une réponse à cette question.

5.2.2 Une variante de l'algorithme de Graham

Nous nous intéressons maintenant à des ordonnancements $(1+\varepsilon)$-approchés, pour tout $\varepsilon > 0$. Nous allons montrer que le prix de la stabilité α-approchée est inférieur à $(1+\varepsilon)$ si α est au moins égal à k, où k est une constante inférieure à $\frac{1}{\varepsilon}$.

Nous considérons pour cela l'algorithme de Graham [42], légèrement modifié :

- Soit k une constante entière.
- Obtenir un ordonnancement optimal pour les k plus grandes tâches, tel que :
  - Une fois que les tâches sont affectées aux machines, elles sont ordonnancées sur chaque machine selon la politique LPT (i.e. chaque machine ordonnance ses tâches de la plus grande à la plus petite).
  - Si deux tâches ont la même longueur, celle qui a le numéro d'identification le plus petit est ordonnancée en premier.
- Ordonnancer les $n-k$ tâches restantes en utilisant l'algorithme de liste LPT.

L'algorithme OPT-LPT(k)

Cet algorithme, noté par la suite OPT-LPT(k), est un schéma d'approximation polynomial (PTAS), et son rapport d'approximation est égal à $1+\varepsilon$, avec $\varepsilon = \frac{1}{2+2\lfloor\frac{k}{2}\rfloor}$, si $m = 2$ et si les k tâches les plus grandes sont ordonnancées de façon optimale [42]. En effet, cet algorithme a le même rapport d'approximation que celui de Graham, qui consiste à ordonnancer de manière optimale les k plus grandes tâches puis à rajouter de manière gloutonne les tâches restantes de la plus grande à la plus petite : la seule différence ici est que l'on réordonne un ordonnancement optimal de manière à respecter les deux conditions indiquées dans l'algorithme. Nous allons maintenant montrer que cet algorithme retourne des équilibres de Nash α-approchés, avec $\alpha < k-2$.

Théorème 5.4 *L'ordonnancement retourné par l'algorithme OPT-LPT(k) est un équilibre de Nash α-approché, avec $\alpha < k-2$.*

Preuve : Montrons que chaque tâche de l'ordonnancement retourné par OPT-LPT(k) ne diminue pas sa date de fin d'un facteur supérieur à $k-2$ en changeant unilatéralement de machine. Les $n-k$ plus petites tâches de l'ordonnancement on été ordonnancées en utilisant la règle LPT, donc elles n'ont pas intérêt à changer de machine. Nous considérons donc le cas des k tâches les plus grandes. Soit $\mathcal{O}$ l'ordonnancement optimal de ces tâches retourné par OPT-LPT(k). Nous considérons maintenant trois cas.

• Dans le premier cas, il y a, dans $\mathcal{O}$, seulement une tâche sur une machine (sans perte de généralité sur P_1), et $k-1$ tâches sur l'autre machine. Puisque cet ordonnancement est optimal, la tâche de P_1 est nécessairement la plus grande tâche de l'ordonnancement, et cet ordonnancement est donc un ordonnancement LPT. Aucune tâche n'a donc intérêt à changer de machine dans ce cas là.

• Considérons maintenant le cas où il y a exactement deux tâches sur une machine (sans perte de généralité sur P_1) dans $\mathcal{O}$. Les $k-2$ tâches restantes sont donc sur P_2.
Comme dans la section précédente, nous noterons x_i la $i^{\text{ème}}$ tâche de P_1, et y_i la $i^{\text{ème}}$ tâche de P_2. Nous allons d'abord montrer qu'aucune tâche ordonnancée sur P_2 n'a intérêt à aller sur P_1. Par construction, on sait que $l_{x_1} + l_{x_2}$ est supérieur ou égal à la somme des longueurs des $k-3$

premières tâches de P_2, $\sum_{j=1}^{k-3} l_{y_j}$. Soit i le plus grand nombre entier tel que $l_{x_1} \geq \sum_{j=1}^{i} l_{y_j}$: les $i+1$ premières tâches de P_2 (i.e. les tâches qui commencent au plus tard à la fin de x_1) n'ont pas intérêt à aller sur P_1, car sinon elles seraient ordonnancées après P_1 et ne diminueraient pas leurs dates de fin. De plus, $l_{x_2} \geq \sum_{j=i+2}^{k-3} l_{y_j}$: les tâches entre y_{i+2} et y_{k-3} (comprises) n'ont donc pas intérêt à changer de machine. De même, y_{k-2} n'a pas intérêt à changer de machine : si elle est plus petite que x_2, alors elle sera ordonnancée sur P_1 après x_2, et ne diminuera pas sa date de fin, car $\mathcal{O}$ est un ordonnancement optimal. Si y_{k-2} est plus grande que x_2, alors y_{k-2} a commencé avant (ou en même temps que) x_2, car sinon $\mathcal{O}$ ne serait pas un ordonnancement optimal. Si elle va sur P_1, y_{k-2} sera ordonnancée après x_1, et ne diminuera donc pas sa date de fin.

La seule tâche qui a éventuellement intérêt à changer de machine est donc x_2. Si x_2 est plus petite que toutes les autres tâches, alors elle n'a pas intérêt à changer de machine. Sinon, puisque $\mathcal{O}$ est un ordonnancement optimal, au moins une tâche de P_2 commence après ou en même temps que x_2. Dans le meilleur des cas, x_2 peut aller en première position sur P_2 : de cette manière, elle commence sur P_2 avant au plus $k-3$ tâches qui commençaient avant elle quand elle était sur P_1. Ces $k-3$ tâches sont plus petites que x_2 : la somme de leurs longueurs, S, est donc plus petite que $(k-3)\, l_{x_2}$. La date de fin de x_2 diminue donc avec ce changement de $S + l_{x_2} < (k-2)\, l_{x_2}$, à l_{x_2}. Donc x_2 est, dans $\mathcal{O}$, α-Nash-approchée, avec $\alpha < k-2$.

• Regardons maintenant le cas où il y a exactement $a < k-2$ tâches sur P_1, et $b < k-2$ tâches sur P_2. Soit t une tâche de P_1 (respectivement de P_2) qui a intérêt à changer de machine. Quand elle change de machine, t dépasse p tâches de P_2 (respectivement de P_1), i.e. elle commence avant p tâches qui commençaient avant t avant le changement. Nous savons que p est inférieur à $k-2$ car il y a moins de $k-2$ tâches sur chaque machine. De plus, ces tâches sont plus petites que t, car sinon t ne les dépasserait pas. Donc t dépasse au plus $k-3$ tâches de longueurs inférieures à l_t, et la date de fin de t diminue d'une valeur inférieure à $(k-2)\, l_t$, à l_t. Donc t est α-Nash-approchée, avec $\alpha < k-2$. $\square$

Nous avons vu que OPT-LPT(k) retourne des équilibres de Nash α-approchés, avec $\alpha < k-2$. Montrons maintenant que nous ne pouvons pas améliorer l'analyse de cet algorithme, car cette borne est atteinte.

Théorème 5.5 *Soit $\varepsilon > 0$, et k un entier supérieur ou égal à 5. L'algorithme OPT-LPT(k) peut retourner des équilibres de Nash α-approchés, avec $\alpha \geq k-2-\varepsilon$.*

Preuve : Soit $\varepsilon' = \frac{\varepsilon}{k-2-\varepsilon}$. Considérons l'instance suivante : une tâche de longueur $k-3-\varepsilon'$, une tâche de longueur $1+\varepsilon'$, et $k-2$ tâches de longueur 1. Le seul ordonnancement optimal pour cet instance est l'ordonnancement dans lequel les tâches de longueur 1 sont sur la même machine (sans perte de généralité, sur P_2), et les deux autres tâches sur l'autre machine. Soit t la tâche de longueur $1+\varepsilon'$: t se termine sur P_1 au temps $k-2$. En changeant de machine, t finirait au temps $1+\varepsilon'$, et diminuerait donc sa date de fin d'un facteur $\frac{k-2}{1+\varepsilon'} = k-2-\varepsilon$. Donc t est $(k-2-\varepsilon)$-Nash-approchée. L'ordonnancement retourné par OPT-LPT(k) sur cette instance est donc un équilibre de Nash α-approché, avec $\alpha \geq k-2-\varepsilon$. $\square$

Nous déduisons du théorème 5.4, et du fait que le rapport d'approximation de OPT-LPT(k) est $\frac{1}{2+2\lfloor\frac{k}{2}\rfloor}$ [42], le résultat suivant :

Corollaire 5.2 *Soit k un entier supérieur ou égal à 5. Le prix de la stabilité α-approchée est au plus de $1+\varepsilon$, où $\varepsilon = \frac{1}{4+2\lfloor \frac{k}{2} \rfloor} < \frac{1}{k}$, pour tout $\alpha \geq k$.*

Remarque : Si l'on souhaite un algorithme $\frac{8}{7}$-approché et qui retourne des solutions aussi stables que possibles, alors LPT_{swap} est meilleur que OPT-LPT(k) : en effet, la solution de LPT_{swap} est trouvée plus rapidement (dans OPT-LPT(k) il y a 64 ordonnancements à comparer, alors qu'il n'y en a que 4 à comparer dans LPT_{swap}), et OPT-LPT(k) retourne des équilibres de Nash 4-approchés (alors que ceux retournés par LPT_{swap} sont 3-approchés). D'un autre côté, OPT-LPT(k) est utile si l'on souhaite des ordonnancements avec un rapport d'approximation plus petit, puisque OPT-LPT(k) est un schéma d'approximation.

Généralisation au cas où il y a $m > 2$ machines : Considérons maintenant l'algorithme OPT-LPT(k) dans le cas où il y a $m > 2$ machines. Cet algorithme retourne également des solutions assez stables dans le cas de m machines. Si $m \geq k$ alors OPT-LPT(k) retourne un ordonnancement LPT, qui est donc un équilibre de Nash. Supposons que $m < k$. De la même manière que dans la preuve du théorème 5.4, les petites tâches, celles qui ne font pas parti des k plus grandes tâches, n'ont pas intérêt à changer de machine : seules les k plus grandes tâches peuvent avoir intérêt à se déplacer. Or il y a au plus $k-m+1$ tâches parmi les k plus grandes tâches sur chaque machine. Chacune de ces tâches, en changeant de machine, ne pourra doubler qu'au plus $k-m+1$ grandes tâches et ces tâches seront de longueur inférieure à elle (car sinon elle ne pourrait pas les doubler) : l'ordonnancement induit par ce changement sera donc un équilibre de Nash $(k-m+1)$-approché, et l'algorithme OPT-LPT(k) retourne des équilibres de Nash $(k-m+1)$-approchés. Le rapport d'approximation de OPT-LPT(k) étant égal à $1 + \frac{1-1/m}{1+\lfloor \frac{k}{m} \rfloor}$ [42], nous en déduisons que le prix de la stabilité α-approchée est inférieur ou égal à $1+\varepsilon$, avec $\varepsilon = \frac{1-1/m}{2+\lfloor \frac{k-1}{m} \rfloor}$, pour tout $\alpha \geq k$.

De même nous pouvons généraliser le théorème 5.5 et montrer que dans le cas de $m > 2$ machines l'algorithme OPT-LPT(k) peut retourner des équilibres de Nash α-approchés avec $\alpha \geq k-m-\varepsilon$ (ceci montre que la borne $(k-m+1)$ ci-dessus est presque atteinte). Pour cela nous considérons l'instance suivante : $m-2$ tâches de longueur $k-m$, une tâche de longueur $k-m-1-\varepsilon$, une tâche t de longueur $1+\varepsilon$, et $k-m$ tâches de longueur 1. Le seul ordonnancement optimal pour cet instance est l'ordonnancement dans lequel chaque tâche de longueur $k-m$ est seule sur sa machine, les tâches de longueur 1 sont sur la même machine, et les deux autres tâches sont sur la machine restante. Comme dans la preuve du théorème 5.5, quand ε tend vers 0, la tâche de longueur $1+\varepsilon$ diminuerait sa date de fin d'un facteur qui tend vers $k-m$ en allant sur la machine sur laquelle se trouvent les tâches de taille 1.

5.2.3 Compromis : stabilité et rapport d'approximation

Nous montrons tout d'abord que si nous voulons un prix de la stabilité α-approché inférieur ou égal à $(1+\varepsilon)$, alors α doit être supérieur à une certaine constante en $\Theta(\varepsilon^{-1/2})$.

Théorème 5.6 *Soit $\varepsilon > 0$ et $k > 0$ tels que $\varepsilon = \frac{1}{k\,(k+1)}$. Pour avoir un prix de la stabilité α-approchée inférieur à $1+\varepsilon$, alors il est nécessaire d'avoir $\alpha \geq k$.*

Preuve : Considérons l'instance suivante : une tâche de longueur $k-1$, une tâche de longueur 1, et $k+1$ tâches de longueur $\frac{k}{k+1}$. Dans un ordonnancement optimal de ces tâches, on a nécessairement les tâches de longueur $\frac{k}{k+1}$ sur la même machine, et les deux autres tâches sur l'autre machine (voir Figure 5.3 *Gauche*). La date de fin d'un tel ordonnancement est $OPT = k$. Cet ordonnancement est un équilibre de Nash k-approché. En effet, la date de fin de la tâche de longueur 1 est k mais cette tâche pourrait être ordonnancée en première position si elle allait sur l'autre machine (car les politiques des machines sont LPT), et sa date de fin serait alors de 1, ce qui est k fois plus petit que sa date de fin actuelle.

FIG. 5.3 – *Gauche :* Ordonnancement optimal pour la date de fin, et équilibre de Nash k-approché (les politiques des machines sont LPT). *Droite :* Ordonnancement approché pour la date de fin.

La figure 5.3 *Droite* montre le second meilleur ordonnancement pour la date de fin : tous les ordonnancements différents de l'ordonnancement optimal (décrit ci-dessus) ont une date de fin supérieure ou égale à la date de fin de cet ordonnancement. Cette date de fin est $k-(\frac{k}{k+1})+1$, et donc le rapport entre cette date de fin et OPT est $\frac{k-(\frac{k}{k+1})+1}{k} = 1+\frac{1}{k}-\frac{1}{k+1} = 1+\frac{1}{k\,(k+1)}$. Ainsi, si nous souhaitons un ordonnancement $(1+\varepsilon)$-approché, avec $\varepsilon < \frac{1}{k\,(k+1)}$, cet ordonnancement ne sera pas un équilibre de Nash α-approché, avec $\alpha < k$. □

De la même manière, nous pouvons montrer le résultat suivant :

Théorème 5.7 *Le prix de la stabilité α-approchée est supérieur ou égal à 1.1 (respectivement $\frac{9}{8}$), pour tout $\alpha \leq \frac{10}{3}$ (respectivement $\alpha \leq \frac{8}{3}$).*

Preuve : Nous utilisons ici la même technique que pour la preuve des théorèmes et 6.12 et 6.13, avec les exemples des figures 5.4 et 5.7. Dans ces exemples l'ordonnancement optimal est $\frac{10}{3}$-Nash approché (respectivement $\frac{8}{3}$-Nash approché) et le deuxième meilleur ordonnancement pour la date de fin est 1.1-approché (respectivement $\frac{9}{8}$-approché). □

FIG. 5.4 – *Gauche :* Ordonnancement optimal. *Droite :* Ordonnancement approché.

FIG. 5.5 – *Gauche :* Ordonnancement optimal. *Droite :* Ordonnancement approché.

Généralisation au cas où il y a $m > 2$ machines : Considérons maintenant le cas où il y a $m > 2$ machines. Il n'est pas possible d'obtenir des résultats meilleurs dans ce cas que lorsque qu'il y a seulement deux machines. En effet, en prenant chaque instance $\mathcal{I}$ utilisée pour montrer un résultat d'impossibilité et en rajoutant $m-2$ tâches ayant comme longueur la date de fin de l'ordonnancement optimal des tâches de $\mathcal{I}$ sur deux machines, alors nous obtenons les mêmes résultats d'impossibilité dans le cas de m machines.

La figure 5.6 illustre le compromis nécessaire entre le rapport d'approximation d'un algorithme, et la stabilité des ordonnancements qu'il retourne, si les politiques des machines sont LPT. La zone colorée en gris foncé illustre les résultats apportés par le théorème 6.13 (pour $2 \leq k \leq 10$), le théorème 6.12, et le théorème 5.7. Chaque point (x, α) appartenant à la zone gris foncé représente un résultat négatif : il signifie qu'il n'existe pas d'algorithme x-approché qui retourne des équilibres de Nash α-approchés. Ceci est vrai pour tout algorithme, et non pas uniquement pour les algorithmes polynomiaux. La zone gris clair illustre les résultats apportés par les théorèmes 5.1 et 5.4. Chaque point (x, α) appartenant à la zone gris clair représente un résultat positif : il signifie qu'il existe un algorithme x-approché qui retourne des équilibres de Nash α-approchés (cet algorithme est soit OPT-LPT(k), soit LPT_{swap}). Si un point (x, α) appartient à la zone blanche, alors cela signifie que nous n'avons pas d'algorithme correspondant à ce point, ni de résultat d'impossibilité disant qu'il n'existe pas d'algorithme x-approché qui retourne des équilibres de Nash α-approchés.

Ces résultats sont valables si la politique de chaque machine est LPT. Nous pouvons nous demander si l'on pourrait obtenir de meilleurs résultats en utilisant comme politique SPT, i.e. si chaque machine ordonnance ses tâches de la plus petite à la plus grande. La section suivante donne une réponse à cette question.

5.3 À propos de la politique SPT

Nous considérons maintenant que nous avons deux machines qui ordonnanent chacune leurs tâches de la plus petite à la plus grande.

Le seul équilibre de Nash, si les politiques sont SPT, correspond à un ordonnancement SPT, i.e. l'ordonnancement que l'on aurait obtenu en utilisant l'algorithme de liste SPT (i.e. en ordonnançant de manière gloutonne chaque tâche, de la plus petite à la plus grande, dès qu'une machine est libre). Le prix de la stabilité, si les politiques des machines sont SPT, est donc égal au rapport d'approximation de l'algorithme de liste SPT, soit 1.5 [42]. Nous allons maintenant voir que le prix

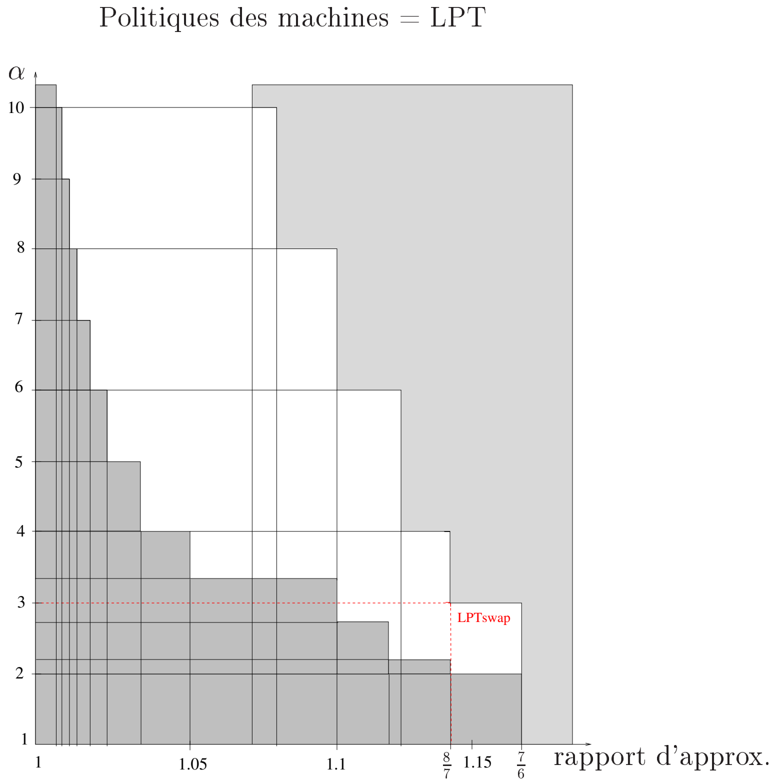


FIG. 5.6 – *Gris clair* (respectivement *Gris foncé*) : Relation entre α et les rapports d'approximation qu'il est possible (respectivement impossible) d'obtenir si l'on souhaite un algorithme qui retourne des équilibres de Nash α-approchés.

de la stabilité α-approché est lui aussi plus grand si les politiques des machines sont SPT plutôt que LPT.

Soit n un (grand) entier positif et pair. Si nous avons une tâche de longueur n et n tâches de longueur 1, alors la date de fin d'un ordonnancement optimal est n : la grande tâche est nécessairement seule sur sa machine, et les petites tâches sont sur l'autre machine. Dans cet ordonnancement optimal, la dernière petite tâche est n-approchée. Au contraire, dans un équilibre de Nash, il y a $n/2$ petites tâches au début de chaque machine, et la grande tâche les suit : le rapport d'approximation est alors de 1.5. Regardons ce qui se passe dans les ordonnancements entre ces deux situations "extrêmes". Dans n'importe quel ordonnancement qui a un rapport d'approximation entre 1 et 1.5, et un prix de la stabilité approchée entre n et 1, il y a x petites tâches avant la grande tâche sur la première machine ($0 < x < n/2$). La figure 5.7 montre un tel ordonnancement.
La date de fin d'un tel ordonnancement est $x+n$, et donc son rapport d'approximation est $1+\frac{x}{n}$. La dernière des petites tâches se termine au temps $n-x$, alors qu'elle pourrait se terminer au temps $x+1$ en changeant de machine : l'ordonnancement est donc un équilibre de Nash ($\frac{n-x}{x+1}$)-approché.

La figure 5.8 montre ce compromis nécessaire entre le rapport d'approximation d'un ordonnancement, et sa stabilité, si les politiques des machines sont SPT. Soit $p = (r, \alpha)$ le point ayant comme abscisse r et comme ordonnée α dans cette figure. Si p se trouve en dessous de la ligne

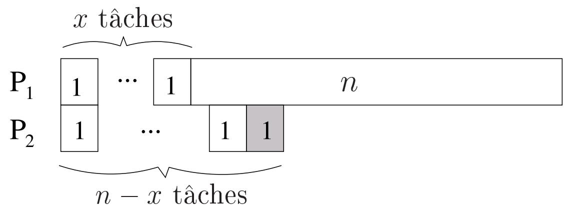


FIG. 5.7 – Ordonnancement $(1+\frac{x}{n})$-approché et équilibre de Nash $(\frac{n-x}{x+1})$-approché.

courbe, alors cela signifie qu'il *n'existe pas* d'algorithme r-approché qui retourne des équilibres de Nash α-approchés si la politique des machines est SPT. Si p est au dessus de l'escalier alors cela signifie qu'il *existe* un algorithme r-approché qui retourne des équilibres de Nash α-approchés si la politique des machines est LPT (cette courbe correspond aux résultats positifs que l'on obtient en utilisant OPT-LPT(k) ou LPT_{swap} et en ayant des machines qui ont comme politique LPT : cette courbe correspond donc à la limite de la zone gris clair dans la figure 5.6). Ainsi, s'il est existe un algorithme r-approché retournant des équilibres de Nash α-approchés avec la politique SPT, alors il existe également un tel algorithme avec la politique LPT. Par contre il existe certaines valeurs de (r, α) (les points entre la ligne courbe et l'escalier) pour lesquelles il existe un algorithme r-approché retournant des équilibres de Nash α-approchés avec la politique LPT mais pas avec la politique SPT. Par exemple, si les politiques sont SPT, alors il n'y a pas d'algorithme qui a un rapport d'approximation inférieur ou égal à 1.1 et qui retourne des équilibres de Nash α-approchés, avec $\alpha \leq 8.8$, alors que, si les politiques sont LPT, $OPT-LPT(8)$ est 1.1-approché et retourne des équilibres de Nash 6-approchés.

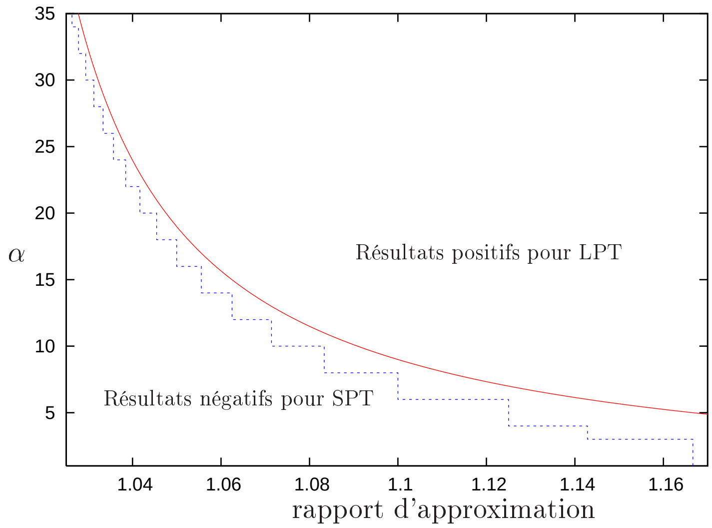


FIG. 5.8 – *En dessous de la courbe* (respectivement *Au dessus de l'escalier*) : Relation entre α et le rapport d'approximation qu'il est impossible (respectivement possible) d'avoir si l'on souhaite un algorithme qui retourne des équilibres de Nash α-approchés, et si les politiques des machines sont SPT (respectivement LPT).

5.4 Une politique probabiliste

Nous nous intéressons dans cette section au cas où la politique des machines n'est plus déterministe mais probabiliste. Le but de chaque tâche n'est alors plus de minimiser sa date de fin mais de minimiser l'espérance de sa date de fin. Nous supposons dans cette section que le nombre de machine est quelconque.
Nous allons considérer la politique qui ordonnance ses tâches selon un ordre aléatoire : chaque machine choisit, dès qu'elle est disponible, aléatoirement une tâche parmi celles demandant à être exécutées (chaque tâche attendant d'être ordonnancée sur cette machine a donc la même probabilité d'être choisie). Nous donnons un algorithme qui propose une affectation des tâches qui induit un ordonnancement optimal (pour la date de fin de l'ordonnancement) et qui est un équilibre de Nash si les tâches, une fois qu'elles ont choisit leur machine, n'ont pas le droit de changer de machine en cours d'exécution. L'espérance de la date de fin des tâches peut varier au cours du temps. Une tâche n'ayant pas intérêt à changer de machine à la date 0 (car elle ne diminuerait pas l'espérance de sa date de fin en changeant de machine à ce moment) peut ainsi avoir intérêt à changer de machine par la suite (par exemple si une grande tâche vient d'être ordonnancée avant elle). Si les tâches sont autorisées à changer de machine au cours de temps, i.e. après la date 0, alors l'algorithme (optimal) que nous proposons retourne des équilibres de Nash 2-approchés.

Cet algorithme, appelé OPT_no_jump, consiste simplement à examiner tous les ordonnancements possibles, et à retourner un ordonnancement optimal tel que aucune tâche ne peut réduire la date de fin de sa machine en changeant de machine. Notons que cet algorithme est exponentiel, ce qui est obligatoire si $P \neq NP$ puisque le problème qui consiste à trouver un ordonnancement optimum pour la date de fin est NP-difficile [40]. Nous montrerons à la fin de cette section qu'il existe également un schéma d'approximation polynomial qui retourne des équilibres de Nash (ou des équilibres de Nash 2-approché dans le cas où les tâches peuvent changer de machine au cours du temps).

- Supposons que n tâches $T = \{1, \ldots, n\}$ doivent être ordonnancées sur m machines. Calculer $Part$, l'ensemble de toutes les partitions de T en m ensembles (i.e. chaque élément de $Part$ est constitué de m sous-ensembles de T distincts deux à deux, et dont l'union est égale à T).
- Soit $\{S_1, \ldots, S_m\} \in Part$. L'ordonnancement associé à cette partition est le suivant : pour chaque $i \in \{1, \ldots, m\}$, les tâches de S_i sont ordonnancées sur la machine P_i, par ordre de longueurs décroissantes.
- Parmi les partitions de $Part$, soit $\{S_1, \ldots, S_m\}$ une partition telle que son ordonnancement associé ait une date de fin minimale, et telle qu'il n'existe pas dans cet ordonnancement de tâche qui commence son exécution après la date de fin d'une machine. Pour chaque $i \in \{1, \ldots, m\}$, l'ensemble des tâches de S_i est affecté à la machine P_i.

L'algorithme OPT_no_jump

La figure 5.9 montre un exemple d'affectation de tâches retournée par OPT_no_jump et un exemple d'affectation de tâches ne pouvant pas être retournée par cet algorithme. Nous supposons que nous avons 4 tâches $\{1, 2, 3, 4\}$, chaque tâche i étant de longueur i. L'algorithme

OPT_no_jump peut retourner l'affectation des tâches $(\{4\}, \{3\}, \{2, 1\})$ correspondant à l'ordonnancement optimal dessiné à gauche de la figure 5.9, mais pas l'affectation $(\{4\}, \{3, 1\}, \{2\})$ correspondant à l'ordonnancement optimal dessiné à droite de la figure 5.9 car dans cet ordonnancement une tâche (la tâche 3, sur P_2) commence à être exécutée après la date de fin d'une machine (P_3).

FIG. 5.9 – *Gauche* : un ordonnancement optimal correspondant à une affectation possible des tâches retournée par OPT_no_jump. *Droite* : ordonnancement optimal qu'il n'est pas possible d'obtenir avec OPT_no_jump.

Nous présentons maintenant un lemme qui sera utile par la suite. Ce lemme énonce un résultat connu, qui a été utilisé notamment dans [55].

Lemme 5.1 *Soient n tâches $1, \ldots, n$ ordonnancées dans un ordre aléatoire sur une même machine. L'espérance de la date de fin de la tâche $t \in \{1, \ldots, n\}$ est égale à $\frac{1}{2}\left(\sum_{j=1, j\neq t}^{n} l_j\right) + l_t$, où l_j est la longueur de la tâche j.*

Preuve : Soit $i \in \{1, \ldots, n\}$. La tâche t a une chance sur n d'être en position i. Soit $S = \sum_{j=1, j\neq t}^{n} l_j$. Calculons l'espérance de la date de fin de t si celle ci est placée en $i^{\text{ème}}$ position. Si t est en position i, alors chaque tâche j a $i-1$ chances sur $n-1$ d'être ordonnancée avant t (ce qui représente une probabilité de $\frac{i-1}{n-1}$). L'espérance de la date de fin de t sachant qu'elle est placée en position i est donc : $\left(\frac{i-1}{n-1}\, S\right) + l_t$. L'espérance de la date de fin de t est donc $\sum_{i=1}^{n} \frac{1}{n}\left(\left(\frac{i-1}{n-1}\, S\right) + l_t\right)$. Ceci est égal à $\frac{1}{n}\left(\frac{S}{n-1}\left(\frac{(n-1)\,n}{2}\right) + n\, l_t\right) = \frac{1}{2}\, S + l_t$. □

Théorème 5.8 *Si les tâches ne peuvent pas changer de machine après le début de l'ordonnancement (i.e. chaque tâche accepte ou refuse son affectation avant le début de l'exécution des tâches), alors l'algorithme OPT_no_jump retourne un équilibre de Nash.*

Preuve : Soit $\{S_1, \ldots, S_m\}$ une affectation des tâches sur les machines telle que proposée par l'algorithme OPT_no_jump, et soit f_{bloc} la date maximum à laquelle toutes les machines sont occupées, i.e. en train d'exécuter une tâche : f_{bloc} est donc la date de fin minimale d'une des machines. Pour chaque $i \in \{1, \ldots, m\}$, soit f_i la somme des longueurs des tâches de S_i. f_{bloc} est donc la valeur minimum de $\{f_1, \ldots, f_m\}$. Au début de l'ordonnancement (à la date 0), les tâches ne savent pas dans quel ordre elles seront exécutées. Considérons une tâche t (de longueur l_t) appartennant à S_i. Si elle ne change pas de machine t sera donc exécutée sur P_i. L'espérance de la date de fin de t est égale à $\frac{f_i - l_t}{2} + l_t$ si elle ne change pas de machine (voir Lemme 5.1). Si elle change unilatéralement de machine, alors l'espérance de sa date de fin sera supérieure ou égale à $\frac{f_{bloc}}{2} + l_t$ car la date de fin de chaque machine est supérieure ou égale à f_{bloc}. Or $f_{bloc} \geq f_i - l_t$ par contruction (car sinon il y aurait eu dans l'ordonnancement associé à $\{S_1, \dots, S_m\}$ une tâche qui commence à être ordonnancée après la date de fin d'une machine). L'espérance de la date de fin de t si elle change unilatéralement de machine est donc supérieure ou égale à $\frac{f_i - l_t}{2} + l_t$, et la tâche t n'a donc pas intérêt à changer de machine. L'algorithme OPT_no_jump retourne donc des équilibres de Nash. $\square$

Théorème 5.9 *L'algorithme OPT_no_jump retourne un équilibre de Nash 2-approché, et ce même si les tâches ont la possibilité de changer de machine après le début de l'ordonnancement.*

Preuve : Soit $\mathcal{S}$ une affectation des tâches sur les machines telle que proposée par l'algorithme OPT_no_jump, et soit f_i la somme des longueurs des tâches placées sur la machine P_i (pour tout $i \in \{1, \dots, m\}$). Supposons qu'une tâche t change unilatéralement de machine, en passant de P_i à P_j, et que la date de fin de la tâche en train d'être exécutée sur P_j au moment du changement est égale à d. L'espérance de la date de fin de t sur P_j est égale à $d + \frac{f_j - d}{2} + l_t > \frac{f_j + l_t}{2}$ (voir Lemme 5.1), alors que l'espérance de la date de fin sur P_i était d'au plus f_i. Or $f_i \leq f_j + l_t$ par construction (car sinon il y aurait eu dans l'ordonnancement associé à l'affectation des tâches retournée par OPT_no_jump une tâche qui commence à être ordonnancée après la date de fin d'une machine). Donc $f_i < 2\left(\frac{f_j + l_t}{2}\right)$ et la tâche t dimininue sa date de fin d'un facteur inférieur à 2 en changeant de machine. $\square$

Notons que si les tâches peuvent changer de machine alors il est possible d'obtenir un équilibre de Nash α-approché où α tend vers 2 : la borne donnée dans le théorème 5.9 est donc atteinte. Considérons pour cela l'ordonnancement suivant : une grande tâche de taille $n-1$ et une petite tâche t de taille 1, affectées à la machine P_1 ; et n tâches de longueur 1 affectées à chacune des autres machines (voir figure 5.10). Si, à la date 0, P_1 commence à exécuter la tâche de taille $n-1$, alors t sait qu'elle finira au temps n. Si elle change alors de machine, l'espérance de sa date de fin sera alors égale à $1 + \frac{n-1}{2} + 1 = \frac{n}{2} + \frac{3}{2}$. Le rapport entre la date de fin de t quand elle ne change pas de machine et l'espérance de la date de fin de t quand elle change de machine machine tend vers 2 quand n tend vers l'infini.

<table>
<tr><td>P_1</td><td colspan="8">$n-1$</td><td>1</td></tr>
<tr><td>P_2</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
</table>

FIG. 5.10 – Ordonnancement dans lequel la tâche en gris a intérêt, dès qu'elle sait qu'elle est ordonnancée en deuxième position, à changer de machine, si les changements sont autorisés au cours du temps.

Enfin, nous avons remarqué que OPT_no_jump n'est pas un algorithme polynomial puisqu'il demande de calculer une solution optimale. Le schéma d'approximation de Graham [42], que nous avons déjà rencontré dans la section 5.2.2, retourne lui aussi des équilibres de Nash si les tâches ne peuvent pas changer de machine au cours du temps, ou un équilibre de Nash 2-approché dans le cas contraire. Nous rappelons que ce schéma d'approximation consiste à ordonnancer de manière optimale les k plus grandes tâches de l'ordonnancement, puis à ordonnancer les tâches restantes, de la plus grande à la plus petite, dès qu'une machine est disponible. Afin que l'affectation des tâches sur les machines retournée par cet algorithme soit un équilibre de Nash (ou un équilibre de Nash 2-approché si les tâches sont mobiles après la date 0), il suffit que l'ordonnancement optimal des k plus grandes tâches soit obtenu avec l'algorithme OPT_no_jump. Les preuves sont alors les mêmes que dans le cas de l'algorithme OPT_no_jump. Par construction, aucune tâche ne peut en effet, en changeant de machine, diminuer la somme des longueurs des tâches placées sur la même machine qu'elle. Ce schéma d'approximation retourne donc en un temps polynomial, pour tout $\varepsilon > 0$, des équilibres de Nash (ou des équilibres de Nash 2-approchés si les tâches sont mobiles au cours du temps) qui ont un rapport d'approximation égal à $(1 + \varepsilon)$.

5.5 Conclusion

Nous avons étudié la relation entre le rapport d'approximation d'un ordonnancement et sa stabilité, et ce selon plusieurs politiques d'ordonnancement des machines. Nous avons dans une première partie considéré le cas où la politique des machines est LPT, et étudié le compromis (nécessaire) entre le rapport d'approximation d'un ordonnancement et sa stabilité. Nous avons ainsi donné des résultats d'impossibilité, et proposé deux algorithmes qui retournent des ordonnancements assez stables par rapport à leur rapport d'approximation. Il reste bien sûr à faire coïncider les bornes inférieures et supérieures que nous avons données, i.e. savoir, pour tout couple (r, α) s'il existe ou non un algorithme r-approché qui retourne des équilibres de Nash α-approchés.

Nous avons montré qu'avec la politique SPT, le prix de la stabilité α-approchée est supérieur au prix de la stabilité α-approchée dans le cas où la politique des machines est LPT : pour tout (r, α), s'il existe un algorithme r-approché retournant des équilibres de Nash α-approché avec la politique SPT, alors il existe également un algorithme r-approché retournant des équilibres de Nash α-approchés avec la politique LPT, alors que l'inverse n'est pas toujours vrai. Savoir s'il existe une politique déterministe meilleure que LPT vis-à-vis du prix de la stabilité approchée reste une question ouverte.

Nous avons considéré le cas où nous autorisons les politiques à ne pas donner à l'avance une date de fin exacte aux tâches mais seulement une espérance de date de fin. Dans ce cas, nous avons donné un algorithme qui, associé à la politique qui ordonnance les tâches selon un ordre aléatoire, retourne des équilibres de Nash si les tâches n'ont pas le droit de changer de machine après le début de l'ordonnancement, ou des équilibres de Nash 2-approchés dans le cas contraire.

Dans le cas des politiques déterministes (LPT et SPT), nous avons principalement étudié le prix de la stabilité approchée en présence de deux machines. Il pourrait être intéressant d'étudier plus en profondeur le cas de $m > 2$ machines, bien que les résultats pour m machines semblent être proches de ceux pour deux machines : les résultats d'impossibilité dans le cas de deux machines sont ainsi les mêmes dans le cas de m machines (il suffit pour cela à chaque contre-exemple basé sur l'instance I de considérer l'instance I et d'y rajouter $m - 2$ tâches de longueurs égale à la date de fin d'un ordonnancement optimal des tâches de I sur deux machines).

Plus généralement, la notion de prix de la stabilité approchée ouvre la voie à de nouveaux travaux étudiant le compromis entre la stabilité d'une solution et sa performance selon un critère donné. Nous avons vu dans le chapitre 1, que l'introduction du prix de la stabilité correspond au fait que les tâches ne sont pas toujours totalement autonomes, mais sont gérées par des protocoles, auxquels elles peuvent ou non obéir : le but est alors d'avoir un protocole qui retourne le meilleur équilibre de Nash vis-à-vis de l'objectif global considéré. Le prix de la stabilité approchée permet d'avoir de meilleures solutions pour cet objectif global, tout en ayant des solutions aussi stables que possible, et en gardant ainsi la confiance des agents, qui savent qu'ils ne tireront pas un grand bénéfice à "dévier" de la solution proposée par le protocole (la stabilité des solutions pouvant être paramétrée avec α). Il semble donc intéressant d'étudier le prix de la stabilité approchée pour d'autres problèmes dans lesquels interviennent des agents autonomes : en ordonnancement, routage, etc.

Chapitre 6

Mécanismes avec véracité garantie pour l'ordonnancement de tâches

Dans ce chapitre, nous nous intéressons à l'ordonnancement de tâches détenues par des agents individualistes sous la contrainte que l'algorithme ne connaît pas directement la longueur des tâches à ordonnancer, celles-ci annonçant elles-mêmes leurs longueurs à l'algorithme qui peut alors retourner un ordonnancement. Le but de chaque tâche (agent) est de se terminer le plus tôt possible, et puisque les agents sont individualistes, ils mentiront sur leur longueur si cela les arrange. Dans ce cas, l'ordonnancement retourné par tout algorithme risque de ne pas être satisfaisant et de ne pas correspondre au rapport d'approximation de l'algorithme, celui-ci étant valable quand les tâches déclarent leurs vraies valeurs. Le but dans ce chapitre est d'étudier le meilleur rapport d'approximation qu'un algorithme à véracité garantie (i.e. dans lequel la meilleure stratégie de chaque agent est de déclarer sa vraie longueur) peut avoir. Une partie des résultats de ce chapitre a été présentée dans [10] et publiée dans [13].

6.1 Introduction

Nous avons vu dans la section 1.2.6 l'importance d'avoir des algorithmes à véracité garantie quand un protocole ne connaît pas dès le départ toutes les données du problèmes mais est face à des agents individualistes qui déclarent eux-mêmes certaines données du problème. Dans ce chapitre, nous nous intéressons à l'ordonnancement de tâches sur des machines parallèles identiques afin de minimiser la date de fin de l'ordonnancement. Nous utilisons les notations habituelles : n tâches, ayant chacune une longueur et un numéro d'identification, doivent être ordonnancées sur m machines. Chaque tâche est détenue par un agent, qui est le seul à connaître la longueur réelle de la tâche. Celui-ci annonce une longueur à l'algorithme chargé d'ordonnancer la tâche, sachant que son but est de minimiser la date de fin de sa tâche. Notre but est de concevoir des algorithmes ayant un bon rapport d'approximation et à véracité garantie, i.e. des algorithmes tels que chaque tâche minimisent sa date de fin en déclarant sa vraie valeur. De cette façon nous pouvons être sûrs du rapport d'approximation de l'algorithme.

Puisque la longueur d'une tâche est connue d'elle même (ou de son agent) uniquement, chaque tâche déclare une longueur qui représente sa longueur. Une tâche ne peut pas déclarer une longueur plus petite que sa longueur réelle car sinon elle ne sera pas exécutée entièrement et n'obtiendra pas le résultat qu'elle souhaite (nous supposons qu'une tâche ne peut pas se diviser en plusieurs tâches ordonnancées indépendamment). Par contre, si une tâche peut diminuer sa date de fin en déclarant une valeur supérieure à sa longueur réelle, alors elle le fera. Un agent peut en effet rajouter des données inutiles à sa tâche pour en augmenter artificiellement la longueur. Nous supposons donc que chaque tâche i de longueur l_i peut déclarer $b_i \geq l_i$. Un algorithme à *véracité garantie* est un algorithme dans lequel aucune tâche n'a intérêt à déclarer une fausse longueur (i.e. une tâche ne peut pas diminuer sa date de fin en déclarant une longueur supérieure à sa longueur réelle).

Nous ne considérons pas de paiements, qui sont souvent nécessaires afin d'avoir un algorithme à véracité garantie (voir chapitre 1). Notre but est de donner des bornes inférieures et supérieures sur le rapport d'approximation qu'un algorithme à véracité garantie sans paiement peut avoir, dans le cas d'un algorithme déterministe ou d'un algorithme probabiliste.

Un algorithme à véracité garantie peut être facilement obtenu en ordonnançant les tâches par ordre de longueurs croissantes. Cet algorithme, qui est l'algorithme de liste SPT, a un rapport d'approximation de $(2 - 1/m)$ [42]. La principale question de ce chapitre est la suivante : *"Existe-t-il un algorithme à véracité garantie avec un meilleur rapport d'approximation pour le problème qui consiste à minimiser la date de fin de l'ordonnancement ?"*

Nous étudions également les performances des mécanismes de coordination à véracité garantie pour le même problème. Les mécanismes de coordinations [30] ont été définis dans la section 1.2.4. Nous rappelons ici brièvement leur définition : dans un mécanisme de coordination, chaque tâche choisit elle même la machine sur laquelle elle va être exécutée et chaque machine a une politique publique qui lui permet de donner un ordre aux tâches qu'elle ordonnance (le problème étant dans ce cas complètement décentralisé, chaque machine connaît uniquement les numéros d'identification et les longueurs – déclarées – des tâches qu'elle ordonnance, et sa politique ne doit donc pas dépendre des tâches qu'elle n'ordonnance pas). Étant donnée la politique des machines et les stratégies des autres tâches, chaque tâche se comporte de façon à minimiser sa date de fin. Nous sommes dans un équilibre de Nash si et seulement si aucune tâche n'a intérêt à changer de stratégie. La stratégie d'une tâche consiste à déclarer sa longueur et à choisir une machine sur laquelle être ordonnancée (voir section 6.4 pour une explication plus formelle). Le *rapport de coordination*, ou *prix de l'anarchie* d'un mécanisme de coordination est égal au rapport maximum entre la date de fin de l'ordonnancement dans un équilibre de Nash obtenu avec ce mécanisme, et la date de fin d'un ordonnancement optimal (obtenu de façon centralisé et dans lequel les tâches déclarent leurs vraies valeurs).

Il est possible, comme il l'a été suggéré dans [30] de légèrement transformer l'algorithme de liste SPT en un mécanisme de coordination à véracité garantie : chaque machine P_i ordonnance ses tâches par longueurs croissantes, et ajoute au tout début de l'ordonnancement (à la date 0) un petit temps d'inactivité de longueur $(i-1)\varepsilon$ multiplié par la longueur de la première tâche ordonnancée sur cette machine. De cette façon, et si ε est assez petit, l'ordonnancement obtenu dans un équilibre de Nash est le même que l'ordonnancement obtenu avec l'algorithme de liste SPT (avec seulement en plus les petits délais au début de l'ordonnancement). Quand ε est inférieur à $1/2\,m$, le rapport de coordination de ce mécanisme est de $2 - 1/m$. Les petits délais sont nécessaires ici uniquement pour éviter d'avoir des équilibres de Nash mixtes. Avec ce mécanisme de coordination, chaque tâche choisit l'unique machine sur laquelle elle minimise sa date de fin.

La question est alors la suivante : *"Existe-t-il un mécanisme de coordination à véracité garantie avec un meilleur rapport de coordination pour le problème qui consiste à minimiser la date de fin de l'ordonnancement ?"*

Dans le cas des algorithmes centralisés, et dans le cas des mécanismes de coordinations, nous considérons les deux modèles d'exécution suivants :

– **Modèle d'exécution fort** : Si la tâche i, de longueur réelle l_i déclare $b_i \geq l_i$, alors sa durée d'exécution restera de l_i (i.e. elle se terminera l_i unités de temps après son départ).
– **Modèle d'exécution souple** : Si la tâche i, de longueur réelle l_i déclare $b_i \geq l_i$, alors sa durée d'exécution sera b_i (i.e. la tâche i – ou son propriétaire – n'aura pas le résultat de son exécution avant b_i unités de temps après le début de l'exécution de i).

6.1.1 Travaux connexes

Nous avons vu dans la section 1.2.6 que le domaine de la conception de mécanismes ("mechanism design") peut être utile pour inciter les agents à déclarer leurs vraies valeurs privées. L'idée principale de ce domaine est de payer les agents pour les inciter à révéler leurs vraies valeurs. La technique la plus célèbre pour obtenir des mécanismes à véracité garantie est probablement le mécanisme Vickrey-Clarke-Groves (VCG) [80, 31, 43]. Cependant, ce mécanisme est à véracité garantie si la valeur de la fonction objectif à optimiser est égale à la somme des valeurs des fonctions objectifs individuelles des agents, et si le mécanisme est capable de calculer une solution optimale du problème. Ceci ne marche pas dans notre cas car d'une part la fonction objectif n'est pas égale à la somme des dates de fin des tâches, et d'autre part notre problème est NP-difficile et il n'est donc pas possible (si $P \neq NP$) de calculer une solution optimale en un temps polynomial. Nous avons également vu que A. Archer et É. Tardos ont introduit dans [18] une méthode qui permet de concevoir des mécanismes à véracité garantie. Cependant cette méthode ne peut pas non plus s'appliquer à notre problème car elle ne marche que lorsque les agents se partagent des charges, et si la valeur secrète des agents est le coût qu'une unité de charge coûte à l'agent. Ces deux approches classiques ne s'appliquant pas à notre problème, nous utiliserons des méthodes ad hoc pour concevoir des mécanismes à véracité garantie.

Les problèmes d'ordonnancement avec des agents autonomes ont mené à de nombreux travaux ces dernières années, en commençant par l'article de N. Nisan et A. Ronen [64] qui a été suivi par de nombreux autres articles [18, 20, 4, 5, 22, 23, 56]. Cependant, tous ces travaux diffèrent des notres car dans leur cas les agents sont les machines, qui déclarent leur vitesse d'exécution, alors qu'ici les agents sont les tâches qui déclarent leurs longueurs. De plus, dans la plupart de ces travaux les mécanismes doivent payer les agents afin de les inciter à révéler leurs vraies valeurs, alors que nous souhaitons concevoir des algorithmes avec véracité garantie sans avoir à payer les tâches.

Il est cependant utile de noter certaines similarités entre certains de ces travaux et les nôtres. Par exemple, dans [22], les auteurs étudient le cas où les agents peuvent mentir dans une seule direction (i.e. ils peuvent soit déclarer une valeur inférieure ou égale à leur vraie valeur, soit déclarer une valeur supérieure ou égale à leur vraie valeur). Cet article traite cependant du contexte des mécanismes avec vérifications (ce qui implique que le mécanisme peut après exécution reconnaître la vraie valeur d'un agent, et le pénaliser s'il n'a pas déclaré sa vraie valeur). Nous considérerons dans la section 6.4.2 un mécanisme de coordination qui est à véracité garantie dans le cas où les longueurs des tâches ne sont pas quelconques mais des puissances d'une constante. L'impact des restrictions que les valeurs que les agents peuvent déclarer a été considéré dans d'autres contextes [69, 46]. D'autres travaux considèrent également des mécanismes à véracité garantie si les valeurs des agents sont restreintes : A. Kovacs a montré dans [56] que l'algorithme de liste LPT est à véracité garantie et 3-approché si les vitesses des machine sont des puissances de 2, et A. Ambrosio et V. Auletta ont montré dans [4] que cet algorithme n'est pas à véracité garantie si les vitesses des machines sont des puissances d'une constante inférieure à 1,78.

Un travail plus proche du notre est celui de Auletta et al. qui ont considéré dans [21] le problème qui consiste à ordonnancer des tâches autonomes avec un algorithme (centralisé) avec véracité garantie. Cependant ce travail diffère du notre car ils considèrent le cas où la date de fin de la tâche est égale à la date de fin de la machine sur laquelle elle se trouve (modèle KP). Les auteurs considèrent de plus qu'une tâche peut mentir dans les deux sens, et qu'il y a des paiements pour inciter les tâches à déclarer leurs vraies valeurs.

Un autre travail connexe est celui de G. Christodoulou et al [30] qui considèrent le modèle que l'on étudie ici (modèle CKN) mais seulement dans le contexte des mécanismes de coordination. Ils proposent différents mécanismes de coordination avec un rapport de coordination meilleur que celui de SPT, mais ces mécanismes ne sont pas à véracité garantie. Les auteurs posent comme question ouverte s'il existe un mécanisme à véracité garantie avec un rapport de coordination meilleur que celui de SPT. Dans [51], les auteurs donnent, pour le même modèle, des mécanismes de coordination dans le cas où les machines ont des vitesses différentes, mais ces mécanismes ne sont pas non plus à véracité garantie.

6.1.2 Résumé des résultats obtenus

Nous donnons des algorithmes avec garantie de véracité ayant un rapport d'approximation meilleur que $2 - \frac{1}{m}$, le rapport d'approximation de SPT, et nous donnons des bornes inférieures sur le meilleur rapport d'approximation qu'un algorithme avec garantie de véracité peut avoir. Nous étudions ces bornes pour des algorithmes déterministes, dans lesquels le but de chaque tâche est de minimiser sa date de fin, et pour des algorithmes probabilistes, dans lesquels le but de chaque tâche est de minimiser l'espérance de sa date de fin. Notre but est de minimiser la date à laquelle toutes les tâches sont exécutées, i.e. la date de fin de l'ordonnancement, et il n'y a pas de paiement pour inciter les tâches à révéler leur vraie valeur. Nous donnons à la fin de cette section un résumé des résultats que nous avons obtenus. Nous étudions le modèle fort d'exécution dans la section 6.2, et dans la section 6.3 nous étudions le modèle souple d'exécution. Dans la section 6.4 nous étudions les performances, en terme de rapport de coordination (ou prix de l'anarchie), qu'un mécanisme de coordination avec véracité garantie peut avoir, pour les deux modèles d'exécution. La section 6.5 conclut ce chapitre.

Les tableaux 6.1 et 6.2 donnent un résumé des résultats apportés dans ce chapitre. Chaque résultat, sauf les deux bornes égales à $2 - \frac{1}{m}$ et associées à une référence bibliographique, est ensuite étudié dans les sections suivantes. Ces tableaux indiquent le rapport d'approximation (respectivement rapport de coordination) qu'un algorithme (respectivement mécanisme de coordination) à véracité garantie peut avoir ou non. Les colonnes “Borne inf.” indiquent une borne inférieure sur ce rapport, i.e. elles indiquent qu'il n'existe pas d'algorithme (ou mécanisme de coordination) à véracité garantie ayant un rapport d'approximation (ou rapport de coordination) inférieur à la valeur indiquée. Les colonnes “Borne sup.” indiquent une borne supérieure sur le rapport d'approximation (ou rapport de coordination) qu'un algorithme (mécanisme de coordination) avec véracité garantie peut avoir : nous donnons alors dans la suite de ce chapitre les algorithmes (mécanismes de coordination) à véracité garantie correspondants à ces bornes.

<table>
<tr><th></th><th colspan="2">Déterministe</th><th colspan="2">Probabiliste</th></tr>
<tr><th></th><th>Borne inf.</th><th>Borne sup.</th><th>Borne inf.</th><th>Borne sup.</th></tr>
<tr><td>algorithme</td><td>$2 - \frac{1}{m}$</td><td>$2 - \frac{1}{m}$ [30]</td><td>$\frac{3}{2} - \frac{1}{2m}$</td><td>$2 - \frac{1}{m+1}\left(\frac{5}{3} + \frac{1}{3m}\right)$</td></tr>
<tr><td>mécanisme de coordination</td><td>$2 - \frac{1}{m}$ (eq. Nash pur)
$\frac{3}{2} - \frac{1}{2m}$ (eq. Nash mixte)</td><td>$2 - \frac{1}{m}$ [30]</td><td>$\frac{3}{2} - \frac{1}{2m}$</td><td>$2 - \frac{1}{m}$</td></tr>
</table>

TAB. 6.1 – Bornes pour m machines dans le modèle fort d'exécution.

<table>
<tr><th></th><th colspan="2">Déterministe</th><th colspan="2">Probabiliste</th></tr>
<tr><th></th><th>Borne inf.</th><th>Borne sup.</th><th>Borne inf.</th><th>Borne sup.</th></tr>
<tr><td>algorithme</td><td>$m = 2 : 1 + \frac{\sqrt{105}-9}{12} > 1,1$
$m \geq 3 : \frac{7}{6} > 1,16$</td><td>$\frac{4}{3} - \frac{1}{3m}$</td><td>1</td><td>1</td></tr>
<tr><td>mécanisme de coordination</td><td>$\frac{1+\sqrt{17}}{4} > 1,28$
(eq. Nash pur)</td><td>$2 - \frac{1}{m}$</td><td>$1 + \frac{\sqrt{13}-3}{4} > 1,15$
(eq. Nash pur)</td><td>$2 - \frac{1}{m}$
(Cas particulier : cf. section 6.4.2)</td></tr>
</table>

TAB. 6.2 – Bornes pour m machines dans le modèle souple d'exécution.

Quelques notations et définitions.

Nous supposons par la suite que nous avons m machines et n tâches $1, \ldots, n$. La longueur réelle de la tâche i est notée l_i. Nous utiliserons par la suite les numéros d'identification pour comparer les tâches de mêmes longueurs : une tâche i déclarant b_i est dite plus grande qu'une tâche j déclarant b_j si et seulement si $b_i > b_j$ ou ($b_i = b_j$ et $i > j$).

Nous dirons qu'un algorithme probabiliste est avec véracité garantie si, pour chaque tâche $i \in \{1, \ldots, n\}$, l'espérance de la date de fin de i quand elle déclare sa vraie longueur est inférieure ou égale à l'espérance de la date de fin de i quand elle déclare une valeur supérieure à sa vraie longueur : un algorithme est à véracité garantie si et seulement si $E_i[l_i] \leq E_i[b_i]$, pour tout i et $b_i \geq l_i$, où $E_i[b_i]$ est l'espérance de la date de fin de i qui déclare b_i. Afin de mesurer la qualité d'un algorithme probabiliste, nous utiliserons l'espérance de son rapport d'approximation.

6.2 Algorithmes centralisés avec véracité garantie dans le modèle fort

6.2.1 Résultats d'inapproximabilité

Nous avons vu que l'algorithme déterministe SPT, qui est $(2 - \frac{1}{m})$-approché, est à véracité garantie. Montrons maintenant qu'il n'existe pas d'algorithme déterministe à véracité garantie avec un meilleur rapport d'approximation.

Théorème 6.1 *Il n'existe pas d'algorithme déterministe à véracité garantie ayant un rapport d'approximation inférieur à* $2 - \frac{1}{m}$.

Preuve : Dans cette preuve, nous donnons une instance pour laquelle il n'existe pas d'algorithme à véracité garantie ayant un rapport d'approximation inférieur à $2 - 1/m$.
Supposons que nous avons un algorithme $\mathcal{A}$ qui a un rapport d'approximation inférieur à $(2-1/m)$, et montrons que $\mathcal{A}$ n'est pas à véracité garantie. Considérons l'instance suivante : m machines et $n = m\,(m-1)+1$ tâches de longueur 1. Soit t la tâche qui a la date de fin maximum, C_t, dans l'ordonnancement retourné par $\mathcal{A}$. La date de fin d'un ordonnancement optimal de ces tâches est m, et donc $C_t \geq m$.

Supposons que la tâche t déclare m au lieu de 1. Montrons que la date de fin de t est alors inférieure à m. Soit OPT la date de fin d'une solution optimale dans laquelle il y a $n-1 = m\,(m-1)$ tâches de longueur 1 et une tâche de longueur m. Nous avons : $OPT = m$. Puisque le rapport d'approximation de l'algorithme $\mathcal{A}$ est inférieur à $(2 - 1/m)$, la date de fin de l'ordonnancement qu'il retourne avec cette instance est inférieure à $(2 - 1/m)\,m = 2\,m - 1$. Dans cet ordonnancement, la tâche de longueur m commence donc avant la date $(m - 1)$. Ainsi, si la tâche t déclare m au lieu de 1, elle commencera avant la date $m - 1$ et sera terminée une unité de temps après le début de son exécution : elle sera donc terminée avant la date m. La tâche t diminue donc sa date de fin en déclarant m à la place de 1, et l'algorithme $\mathcal{A}$ n'est pas à véracité garantie. □

Notons que nous pouvons généraliser ce résultat dans le cas où les machines ont des vitesses différentes : nous avons m machines $P_1, \ldots, P_m$, telles que la machine P_i a une vitesse égale à v_i, $v_1 = 1$, et $v_1 \leq \cdots \leq v_m$. Nous obtenons alors qu'il n'existe pas d'algorithme déterministe à véracité garantie qui a un rapport d'approximation inférieur à $2 - \frac{v_m}{\sum_{i=1}^{m} v_i}$ (la preuve est en annexe).

Théorème 6.2 *Il n'existe pas d'algorithme probabiliste à véracité garantie ayant un rapport d'approximation inférieur à* $\frac{3}{2} - \frac{1}{2m}$.

Preuve : Soit $\varepsilon > 0$. Supposons que nous avons un algorithme à véracité garantie $\mathcal{A}$ dont l'espérance de son rapport d'approximation est $\big(3/2 - 1/(2m) - \varepsilon\big)$.

Considérons l'instance suivante : m machines et $x\,m\,(m-1)+m$ tâches de longueur 1, où x est un entier positif tel que $x > 1/(2\,\varepsilon\,m) - 1/(2\,\varepsilon\,m^2) - 1/m$. Puisque m machines sont disponibles, la date de fin d'un ordonnancement optimal est $x\,(m-1)+1$. Quelque soit l'algorithme probabiliste utilisé ($\mathcal{A}$ y compris), il existe une tâche t dont l'espérance de sa date de fin est supérieure ou égale à $(x\,(m-1))/2+1$. Puisque l'algorithme $\mathcal{A}$ est à véracité garantie, la tâche t ne doit pas diminuer l'espérance de sa date de fin en déclarant $x\,m+1$ au lieu de 1. Supposons que t mente (unilatéralement) et déclare $x\,m+1$ à la place de 1. Du point de vue de l'algorithme $\mathcal{A}$, il existe donc $x\,m\,(m-1)+m-1$ tâches de longueur 1 et une tâche de longueur $x\,m+1$. La date de fin d'un ordonnancement optimal est alors de $x\,m+1$ et l'espérance de la date de fin de l'ordonnancement retourné par $\mathcal{A}$ est inférieure ou égale à $(3/2-1/(2m)-\varepsilon)(x\,m+1)$. L'espérance de la date de fin de la tâche de longueur $x\,m+1$ est donc aussi majorée par $(3/2-1/(2m)-\varepsilon)(x\,m+1)$. Puisque t a artificiellement augmenté sa longueur de $x\,m$ unités de temps, l'espérance de sa date de fin réelle est inférieure ou égale à $(3/2-1/(2m)-\varepsilon)(x\,m+1)-x\,m$. Avec $x > 1/(2\varepsilon m)-1/(2\varepsilon m^2)-1/m$, l'espérance de la date de fin de t est strictement inférieure à $x\,(m-1)/2+1$ quand t déclare $x\,m+1$, et supérieure ou égale à $(x\,(m-1)+2)/2$ quand elle déclare sa vraie valeur. Ceci contredit le fait que l'algorithme $\mathcal{A}$ soit à véracité garantie. $\square$

Nous pouvons également généraliser ce résultat dans le cas où les machines ont des vitesses différentes : nous avons m machines $P_1,\dots,P_m$, telles que la machine P_i a une vitesse égale à v_i, $v_1=1$, et $v_1 \leq \dots \leq v_m$. Nous obtenons alors qu'il n'existe pas d'algorithme à véracité garantie qui a un rapport d'approximation inférieur à $\frac{3}{2}-\frac{v_m}{2\sum_{i=1}^m v_i}$.

Nous donnons dans la section suivante un algorithme probabiliste et à véracité garantie. Cet algorithme est obtenu en utilisant un algorithme déterministe et à véracité garantie, qui sera utilisé avec une certaine probabilité p, et l'algorithme de liste LPT (qui n'est pas à véracité garantie mais possède un bon rapport d'approximation), avec une probabilité $(1-p)$.

6.2.2 L'algorithme $LPT \oplus DSPT$

Puisque l'algorithme de liste SPT est à véracité garantie et que l'algorithme de liste LPT a un rapport d'approximation meilleur que SPT, il est naturel de se demander si l'algorithme probabiliste qui retourne un ordonnancement SPT avec une certaine probabilité p et un ordonnancement LPT avec une probabilité $1-p$, est à véracité garantie pour certaines valeurs de p $(0 \leq p < 1)$. Nous pouvons facilement remarquer que ceci n'est pas le cas, quelque soit $p < 1$. En effet, considérons l'instance suivante sur deux machines : une tâche t de longueur 1, une tâche de longueur 2, et une tâche de longueur 3. Si t, la tâche de longueur 1, déclare sa vraie longueur, elle sera ordonnancée en première position dans l'ordonnancement SPT (et finira donc au temps 1), et sera ordonnancée dans l'ordonnancement LPT après la tâche de longueur 2 (elle finira alors au temps 3). L'espérance de la date de fin de t qui déclare sa vrai longueur est donc $p+3\,(1-p)=3-2\,p$. Si t déclare 2.5, au lieu de sa vraie longueur 1, alors elle sera ordonnancée en première position dans chacun des ordonnancements SPT et LPT. Dans chacun des cas elle se terminera donc 1 unité de temps après son départ, c'est à dire au temps 1. L'espérance de sa date de fin quand elle déclare 2.5 est donc $1 < 3-2\,p$. Puisque l'espérance de la date de fin de t est inférieure quand elle déclare une fausse longueur plutôt que sa vraie longueur, l'algorithme qui retourne avec une probabilité $p < 1$ un ordonnancement SPT et avec une probabilité $1-p$ un ordonnancement LPT n'est pas à véracité garantie. Nous allons maintenant voir qu'en modifiant légèrement l'algorithme de liste SPT, il est possible d'obtenir un algorithme probabiliste de ce type à véracité garantie.

Considérons l'algorithme DSPT (pour "Delayed SPT"), qui retourne un ordonnancement SPT dans lequel un délai (temps d'inactivité) peut éventuellement précéder chaque tâche. Nous montrerons ensuite que cet algorithme est à véracité garantie, et a un rapport d'approximation égal à $2 - \frac{1}{m}$.

> Soit $\{1, 2, \ldots, n\}$ un ensemble de n tâches à ordonnancer sur $m \geq 2$ machines identiques $\{P_1, P_2, \ldots, P_m\}$. Supposons que $l_1 \leq l_2 \leq \cdots \leq l_n$.
> Les tâches sont ordonnancées alternativement sur $P_1, P_2, \ldots, P_m$, par ordre de longueurs croissantes, et la tâche $i + 1$ commence à être exécutée quand exactement $\frac{1}{m}$ de la tâche i a été exécutée.
> Ainsi la tâche 1 commence à être exécutée sur P_1 à la date 0, la tâche 2 est ordonnancée sur P_2 à la date $\frac{l_1}{m}$, la tâche 3 est ordonnancée sur $P_{3 \bmod m}$ (i.e. sur P_1 si $m = 2$, sur P_3 sinon) quand $\frac{1}{m}$ de la tâche 2 a été exécutée, i.e. au temps $\frac{l_1}{m} + \frac{l_2}{m}$, et ainsi de suite...

L'algorithme DSPT

L'ordonnancement retourné par l'algorithme DSPT sera appelé un ordonnancement DSPT par la suite. La figure 6.1 montre un exemple d'un tel ordonnancement lorsque $m = 3$.

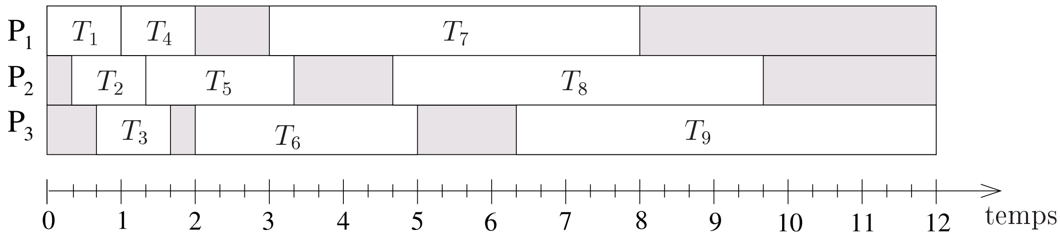


FIG. 6.1 – Un ordonnancement DSPT. T_i représente la tâche i.

Le lemme suivant montre qu'un ordonnancement DSPT est toujours réalisable.

Lemme 6.1 *Supposons que nous avons n tâches $\{1, \ldots, n\}$ à ordonnancer sur m machines $\{P_1, \ldots, P_m\}$. Soit $délai(i)$ le temps d'inactivité ajouté par l'algorithme DSPT avant la tâche i : si $i \leq m$, $délai(i)$ est égal à la date de début d'exécution de la tâche i, sinon $délai(i)$ est égal à la date de début d'exécution de i moins la date de fin de $i - m$. Pour chaque $i \in \{1, \ldots, n\}$, nous avons $délai(i) \geq 0$.*

Preuve : L'algorithme DSPT ajoute, à chaque étape $i \in \{1, \ldots, n\}$, la tâche i sur la machine $P_{i \bmod m}$. Montrons que le temps d'inactivité (ou délai) avant chaque tâche n'est pas négatif. Il est trivial que $délai(i) \geq 0$ pour les m tâches ordonnancées en première position. Soit i une tâche qui n'est pas ordonnancée en première position (i.e. i est ordonnancée après la tâche $(i - m)$). La tâche i commence à être exécutée à exactement $\frac{1}{m}$ de l'exécution de la tâche $i - 1$, qui commence à être exécutée à exactement $\frac{1}{m}$ de l'exécution de $i - 2$, ... , la tâche $(i - m + 1)$ qui commence à être exécutée à exactement $\frac{1}{m}$ de l'exécution de $i - m$. Ainsi la tâche i commence à être exécutée $\frac{1}{m}(l_{i-1} + l_{i-2} + \cdots + l_{i-m})$ unités de temps après le début de la tâche $i - m$. Puisque $l_{i-1} \geq l_{i-2} \geq \cdots \geq l_{i-m}$, $\frac{1}{m}(l_{i-1} + l_{i-2} + \cdots + l_{i-m}) \geq l_{i-m}$, et donc $délai(i) \geq 0$. □

Théorème 6.3 *L'algorithme $DSPT$ est $2 - \frac{1}{m}$-approché : la date de fin d'un ordonnancement $DSPT$ est inférieure ou égale à $(2 - \frac{1}{m})\, OPT$, où OPT est la date de fin d'un ordonnancent optimal de la même instance.*

Preuve : Soit n tâches $1, \ldots, n$, telles que $l_1 \leq \cdots \leq l_n$, à ordonnancer sur m machines. Chaque tâche i commence à être exécutée quand exactement $\frac{1}{m}$ de $i-1$ a été exécutée. Donc, si $n \leq m$, la date de fin de l'ordonnancement DSPT est : $\frac{1}{m}\,(l_1 + \cdots + l_{n-1}) + l_n \leq \frac{1}{m}\,(n-1)\; l_n + l_n \leq \frac{(2m-1)\, l_n}{m} \leq (2 - \frac{1}{m})\, l_n \leq (2 - \frac{1}{m})\, OPT$, car $l_n \leq OPT$.

Considérons maintenant le cas où $n > m$. Soit $i \in \{m+1, \ldots, n\}$. La tâche i commence à être exécutée quand $\frac{1}{m}$ de $i-1$ est exécutée, et la tâche $i-1$ a elle même commencé son exécution quand exactement $\frac{1}{m}$ de la tâche $i-2$ était exécuté, …, et la tâche $(i-m)+1$ a commencé a être exécutée quand $\frac{1}{m}$ de $i-m$ était exécutée. Le temps d'inactivité entre les tâches i et $i-m$ est donc $délai(i) = \frac{1}{m}\,(l_{i-m} + l_{i-m+1} + \cdots + l_{i-1}) - l_{i-m}$.
Soit $i \in \{2, \ldots, m\}$. Le temps d'inactivité avant la tâche i est égal à $délai(i) = \frac{1}{m}\,(l_1 + \cdots + l_{i-1})$. Il n'y a pas de temps d'inactivité avant la tâche 1, qui commence à être exécutée à la date 0. Ainsi, la somme des temps d'inactivité présents avant les tâches est égale à $\sum_{i=2}^{n} délai(i) = \frac{1}{m}((m-1)\, l_{n-m+1} + (m-2)\, l_{n-m+2} + \cdots + l_{n-1})$.

Soit $j \in \{n-m+1, \ldots, n-1\}$. Soit $inactivité\_fin(j)$ le temps d'inactivité à la fin de l'ordonnancement sur la machine sur laquelle se trouve la tâche T_j (i.e. le temps d'inactivité après la fin de la tâche j et avant la fin de n) : $inactivité\_fin(j) = l_{j+1} - \frac{m-1}{m}\, l_j + inactivité\_fin(j+1)$, où $inactivité\_fin(n) = 0$. Donc la somme des temps d'inactivité entre la fin de la dernière tâche de chaque machine et la fin de l'ordonnancement est égale à $\sum_{j=n-m+1}^{n-1} inactivité\_fin(j) = (m-1)\left(l_n - \frac{m-1}{m}\, l_{n-1}\right) + (m-2)\left(l_{n-1} - \frac{m-1}{m}\, l_{n-2}\right) + \cdots + \left(l_{n-m+2} - \frac{m-1}{m}\, l_{n-m+1}\right)$.

La somme des temps d'inactivité sur les machines, du début de l'ordonnancement à la fin de l'ordonnancement, est égale à la somme des délais avant chaque tâche de l'ordonnancement, plus la somme des temps d'inactivité entre la fin de la dernière tâche de chaque machine et la fin de l'ordonnancement. Cette somme est égale à $\sum_{i=2}^{n} délai(i) + \sum_{j=n-m+1}^{n-1} inactivité\_fin(j) = \frac{1}{m}\,((m-1)\, l_{n-m+1} + (m-2)\, l_{n-m+2} + \cdots + l_{n-1}) + (m-1)\left(l_n - \frac{m-1}{m}\, l_{n-1}\right) + (m-2)\left(l_{n-1} - \frac{m-1}{m}\, l_{n-2}\right) + \cdots + \left(l_{n-m+2} - \frac{m-1}{m}\, l_{n-m+1}\right) = (m-1)\, l_n$.

Soit ξ la date de fin d'un ordonnancement DSPT. ξ est égale à la somme des longueurs des tâches plus la somme des temps d'inactivité, divisée par m : $\xi = \frac{\left(\sum_{i=1}^{n} l_i\right) + (m-1)\, l_n}{m} = \frac{\sum_{i=1}^{n} l_i}{m} + \frac{(m-1)\, l_n}{m}$. Comme $\frac{\sum_{i=1}^{n} l_i}{m} \leq OPT$ et $l_n \leq OPT$, nous avons : $\xi \leq (2 - \frac{1}{m})\, OPT$. □

Considérons maintenant l'algorithme $LPT \oplus DSPT$:

> Soit m le nombre de machines.
> Avec une probabilité $\frac{m}{m+1}$, retourner un ordonnancement DSPT ; et avec une probabilité $\frac{1}{m+1}$, retourner un ordonnancement LPT.

L'algorithme $LPT \oplus DSPT$

Théorème 6.4 *L'espérance du rapport d'approximation de $LPT \oplus DSPT$ est égale à $2 - \frac{1}{m+1}\left(\frac{5}{3} + \frac{1}{3\,m}\right)$.*

Preuve : Le rapport d'approximation d'un ordonnancement DSPT est $2 - \frac{1}{m}$ (voir théorème 6.3), et le rapport d'approximation d'un ordonnancement LPT est $\frac{4}{3} - \frac{1}{3m}$ (voir [42]). L'espérance du rapport d'approximation de $LPT \oplus DSPT$ est donc $\frac{m}{m+1}\left(2 - \frac{1}{m}\right) + \frac{1}{m+1}\left(\frac{4}{3} - \frac{1}{3m}\right) = \frac{1}{m+1}\left(2m - 1 + \frac{4}{3} - \frac{1}{3m}\right) = \frac{1}{m+1}\left(2(m+1) - \frac{5}{3} - \frac{1}{3m}\right) = 2 - \frac{1}{m+1}\left(\frac{5}{3} + \frac{1}{3m}\right)$. □

6.2.3 Véracité garantie de l'algorithme $LPT \oplus DSPT$

Théorème 6.5 *L'algorithme $LPT \oplus DSPT$ est à véracité garantie.*

Preuve : Supposons que nous avons n tâches $\{1, \ldots, n\}$ ordonnées par longueurs croissantes, à ordonnancer sur m machines. Montrons que la tâche i n'a pas intérêt à déclarer une longueur supérieure à sa longueur réelle l_i. Supposons que la tâche i déclare $b > l_i$, et que, en déclarant b, la tâche i devienne alors supérieure ou égale aux tâches $1, \ldots, x$, et inférieure à la tâche $x + 1$ (nous rappelons qu'une tâche a est supérieure à une tâche b si et seulement si $l_a > l_b$, ou $l_a = l_b$ et $a < b$). Dans l'ordonnancement LPT, les tâches $x + 1$ à n sont ordonnancées de la même façon, que la tâche i déclare l_i ou b. En déclarant b, la tâche i peut dans cet ordonnancement, au mieux, commencer $(l_{i+1} + \cdots + l_x)$ unités de temps avant la date à laquelle elle aurait commencée en déclarant l_i. Grâce à l'ordonnancement LPT, l'espérance de la date de fin de la tâche i dans $LPT \oplus DSPT$ diminue donc d'au plus $\frac{1}{m+1}(l_{i+1} + \cdots + l_x)$ unités de temps quand T_i déclare b au lieu de l_i.
Au contraire, en déclarant b au lieu de l_i, la tâche i finira plus tard dans l'ordonnancement DSPT : dans cet ordonnancement, les tâches $i + 1$ à x commenceront avant i. Puisqu'une tâche j commence à être exécutée quand son prédécesseur $j - 1$ en est à $\frac{1}{m}$ de son exécution, en déclarant b, la tâche i commencera $\frac{1}{m}(l_{i+1} + \cdots + l_x)$ unités de temps après la date à laquelle elle aurait commencée en déclarant l_i. L'ordonnancement SPT fait donc augmenter l'espérance de la date de fin de la tâche T_i dans $LPT \oplus DSPT$ de $\frac{m}{m+1}(\frac{1}{m}(l_{i+1} + \cdots + l_x)) = \frac{1}{m+1}(l_{i+1} + \cdots + l_x)$.
Ainsi, globalement, l'espérance de la date de fin de la tâche i ne peut pas diminuer quand cette tâche déclare une valeur plus grande que sa vraie longueur l_i, et l'algorithme $LPT \oplus DSPT$ est donc à véracité garantie. □

Nous remarquons que nous ne pouvons pas obtenir de meilleur rapport d'approximation en choisissant les probabilités pour DSPT et LPT d'une autre façon. En effet, si nous avons $m \geq 2$ machines, $m - 1$ tâches de longueur m et m tâches de longueur 1, alors, en déclarant $1 + \varepsilon$ (où ε est une valeur positive négligeable), la tâche de longueur 1 qui est la dernière dans l'ordonnancement LPT gagnera $(m - 1)$ unités de temps dans l'ordonnancement LPT, alors qu'elle perdra $\frac{1}{m}(m - 1)$ unités de temps dans l'ordonnancement DSPT.

Si par exemple nous avons deux machines ($m = 2$), le rapport d'approximation de $LPT \oplus DSPT$ est de $\frac{25}{18} < 1,39$, alors que celui de l'algorithme SPT est de $1,5$. Notons que cet algorithme est à véracité garantie même si les tâches peuvent déclarer n'importe quelle valeur, et son rapport d'approximation est meilleur que celui de $SPT \oplus_p SL$ qui sera présenté dans la section 6.4.2 (cependant $LPT \oplus DSPT$, contrairement à $SPT \oplus_p SL$, n'est pas un mécanisme de coordination car chaque machine doit connaître les tâches ordonnancées sur les autres machines).

Nous remarquons également que, puisque le rapport d'approximation d'un ordonnancement DSPT est $2 - \frac{1}{m}$ (comme SPT), et que le rapport d'approximation d'un ordonnancement LPT est $\frac{4}{3} - \frac{1}{3m}$, l'ordonnancement retourné par $LPT \oplus DSPT$ est, dans le pire des cas, $2 - \frac{1}{m}$-approché, ce qui n'est pas pire que le rapport d'approximation d'un ordonnancement SPT.

6.3 Algorithmes centralisés avec véracité garantie dans le modèle souple

6.3.1 Résultats d'inapproximabilité

Théorème 6.6 *Si nous avons une instance de tâches à ordonnancer sur deux machines, il n'existe pas d'algorithme déterministe à véracité garantie ayant un rapport d'approximation inférieur à* $1 + \frac{\sqrt{105}-9}{12} \approx 1,1039$.

Preuve : Pour des raisons de lisibilité, nous prouvons ce théorème avec un rapport d'approximation de $1,1$. À la fin de la preuve, nous indiquons comment remplacer certaines valeurs numériques par d'autres afin d'obtenir un rapport égal à $1 + (\sqrt{105} - 9)/12$.

Supposons que nous avons un algorithme à véracité garantie $\mathcal{A}$ ayant un rapport d'approximation $\rho < 1,1$. Soit I l'instance suivante : une tâche de longueur 5, une tâche de longueur 4, et trois tâches de longueur 3. La date de fin d'un ordonnancement optimal de ces tâches est 9. Lorsqu'il a I en entrée, l'algorithme $\mathcal{A}$ retourne un ordonnancement σ dont la date de fin est strictement inférieure à 9.9. Puisque $\mathcal{A}$ est un algorithme déterministe, il doit donc exécuter les trois tâches de longueur 3 sur la même machine. Deux de ces tâches ont alors une date de fin supérieure ou égale à 6. Soit I' l'instance suivante : deux tâches de longueur 5, une tâche de longueur 4, et deux tâches de longueur 3. La date de fin d'un ordonnancement optimal de ces tâches est 10. Lorsqu'il a I' en entrée, l'algorithme $\mathcal{A}$ retourne un ordonnancement σ' dont la date de fin est strictement inférieure à 11. Puisque $\mathcal{A}$ est un algorithme déterministe, il doit exécuter les deux tâches de longueur 5 sur la même machine. Ainsi, l'une des tâches de longueur 5 (la première à être ordonnancée) a une date de fin inférieure à 6.

Puisque $\mathcal{A}$ est à véracité garantie, aucune tâche ne doit pouvoir diminuer sa date de fin en déclarant une longueur supérieure à sa vraie longueur. Ainsi, parmi les deux tâches de longueur 3 qui se terminent après le date 6 dans σ, aucune ne peut déclarer 5 et être la tâche qui se termine avant le temps 6 dans σ'. Nous montrons maintenant que $\mathcal{A}$ ne peut pas éviter cette situation, et que cet algorithme n'est donc pas à véracité garantie.

Soit $ID = \{a, b, c, d\}$ un ensemble de quatre numéros d'identification distincts (*id* signifie *numéro d'identification* dans la suite de cette preuve). Pour chaque $x \in ID$, nous appelons I_x l'instance I dans laquelle les ids sont affectés de la façon suivante : la tâche de longueur 5 a l'id x et les trois tâches de longueur 3 ont un id dans $ID \setminus \{x\}$ (la tâche de longueur 4 a un id $e \notin ID$). Soit σ_x l'ordonnancement retourné par $\mathcal{A}$ ayant I_x en entrée. Soit $S = \{I_x \mid x \in ID\}$. Nous avons $|S| = 4$.

Pour chaque couple $\{x, y\} \subset ID$, nous appelons $I'_{x,y}$ l'instance I' dans laquelle les ids sont affectés de la façon suivante : les deux tâches de longueur 5 ont les ids x et y, et les deux tâches de longueur 3 prennent leurs ids dans $ID \setminus \{x, y\}$ (la tâche de longueur 4 a toujours l'id e). Soit

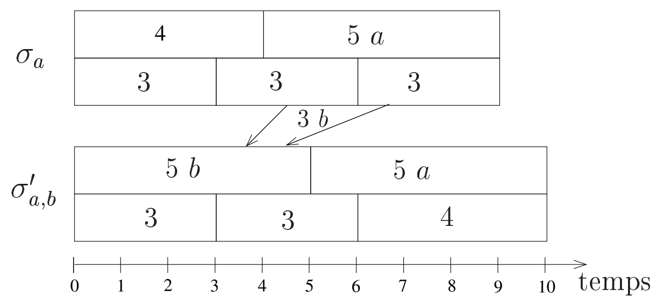


FIG. 6.2 – Illustration de la preuve du théorème 6.6. Le chiffre à l'intérieur de chaque tâche indique sa longueur et la lettre éventuelle son numéro d'identification. En haut : l'ordonnancement σ_a, et en bas l'ordonnancement $\sigma'_{a,b}$. La tâche de longueur 3 et de numéro d'identification b a intérêt à déclarer une longueur égale à 5.

$\sigma'_{x,y}$ l'ordonnancement retourné par $\mathcal{A}$ ayant $I'_{x,y}$ en entrée. Soit $S' = \{I'_{x,y} \mid \{x, y\} \subset ID\}$. Nous avons $|S'| = 6$.

Soit $f : S' \to S$ une fonction telle que $f(I'_{x,y}) = I_x$ si la tâche ayant x comme id est ordonnancée après la tâche d'id y dans $\sigma'_{x,y}$; sinon $f(I'_{x,y}) = I_y$. Puisque $|S'| > |S|$, il existe un couple d'instances de S' qui ont toutes deux la même image dans S avec la fonction f. Sans perte de généralité, nous supposons que ces instances sont $I'_{a,b}$ et $I'_{a,c}$ et que $f(I'_{a,b}) = f(I'_{a,c}) = I_a$. Puisque I_a contient trois tâches de longueur 3 (ayant comme ids b, c et d) qui sont ordonnancées sur la même machine dans σ_a, deux d'entre elles ont une date de fin supérieure ou égale à 6. Les ids de ces tâches sont dans $\{b, c, d\}$ et ont donc une intersection non vide avec $\{b, c\}$. Sans perte de généralité, supposons que la tâche b de longueur 3 a une date de fin supérieure ou égale à 6 dans σ_a. Puisque $f(I'_{a,b}) = I_a$, si la tâche d'id b déclare 5 au lieu de 3, elle sera exécutée avant a dans $\sigma'_{a,b}$. Comme nous l'avons observé précédemment, la date de fin de b sera alors strictement inférieure à 6. Ainsi, $\mathcal{A}$ n'est pas un algorithme à véracité garantie, et il n'existe pas d'algorithme à véracité garantie et ρ-approché avec $\rho < 1,1$.

En utilisant la même technique avec des instances légèrement modifiées, nous pouvons améliorer légèrement cette borne inférieure sur le meilleur rapport d'approximation qu'un algorithme à véracité garantie peut avoir. Ainsi, il n'existe pas d'algorithme à véracité garantie et ayant un rapport d'approximation de $\rho < 1 + (\sqrt{105} - 9)/12 \approx 1,1039$. Pour montrer ce résultat, l'instance I contient trois tâches de longueur 2, une tâche de longueur $2 + 2\varepsilon$ et une tâche de longueur $3 + \varepsilon$. L'instance I' contient deux tâches de longueur 2, une tâche de longueur $2 + 2\varepsilon$ et deux tâches de longueur $3 + \varepsilon$. La borne est alors obtenue quand $\varepsilon = (\sqrt{105} - 9)/4$. □

Théorème 6.7 *Si nous avons une instance de tâches à ordonnancer sur $m \geq 3$ machines, il n'existe pas d'algorithme déterministe à véracité garantie ayant un rapport d'approximation inférieur à $\frac{7}{6} = 1,16667$.*

Preuve : Supposons que nous avons un algorithme $\mathcal{A}$ avec véracité garantie ayant un rapport d'approximation $\rho < 7/6$. Soit I l'instance suivante : une tâche de longueur 3 et $3\,m - 2$ tâches de longueur 2. Soit σ l'ordonnancement retourné par $\mathcal{A}$ ayant comme entrée I. Puisque seulement m machines sont disponibles, au plus m tâches de longueur 2 se terminent avant 4 unités de temps, et donc au moins $2\,m-2$ tâches de longueur 2 ont une date de fin supérieure ou égale à 4.

Considérons maintenant l'instance suivante I' : deux tâches de longueur 3, et $3\,m-3$ tâches de longueur 2. La date de fin d'un ordonnancement optimal de ces tâches est 6. Quand il a l'instance I' en entrée, l'algorithme $\mathcal{A}$ retourne un ordonnancement σ' dont la date de fin est strictement inférieure à 7. Ainsi $\mathcal{A}$ doit nécessairement exécuter les deux tâches de longueur 3 sur la même machine, et la tâche de longueur 3 qui est ordonnancée en première se termine strictement avant 4 unités de temps dans σ'.

Puisque $\mathcal{A}$ est à véracité garantie, aucune tâche ne doit pouvoir diminuer sa date de fin en déclarant une longueur plus grande que sa longueur réelle. En particulier, parmi les tâches de longueur 2 qui se terminent au plus tôt à la date 4 dans σ, aucune ne doit pouvoir, en déclarant 3, être la tâche de longueur 3 qui se termine strictement avant la date 4 dans σ'. Nous montrons maintenant que $\mathcal{A}$ ne peut éviter une telle situation et qu'il n'est donc pas à véracité garantie.

Soit ID un ensemble de $3\,m-1$ numéros d'identification (ids) distincts. Pour chaque sous-ensemble $\{a,b\} \subset ID$, nous appelons $I'_{a,b}$ l'instance I' dans laquelle les ids sont affectés de la façon suivante : les deux tâches de longueur 3 ont les ids a et b, et les $3\,m-3$ tâches de longueur 2 ont chacune un id de $ID-\{a,b\}$. Soit $\sigma'_{a,b}$ l'ordonnancement retourné par $\mathcal{A}$ ayant $I'_{a,b}$ en entrée.

ID contient nécessairement $m+2$ ids distincts $i_1,\ldots,i_{m+2}$ tels que : $\forall x \in \{1,\ldots,m+1\}$, la tâche d'id i_x est ordonnancée avant celle d'id i_{m+2} dans $\sigma'_{i_x,i_{m+2}}$.

Nous considérons maintenant l'instance $I_{i_{m+2}}$ qui est égale à l'instance I dans laquelle la tâche de longueur 3 a l'id i_{m+2} et les $3\,m-2$ tâches de longueur 2 prennent leur id dans $ID\backslash\{i_{m+2}\}$. Soit $\sigma_{i_{m+2}}$ l'ordonnancement retourné par $\mathcal{A}$ ayant en entrée $I_{i_{m+2}}$. Il existe au moins une tâche ayant comme id i_y tel que $y \in \{1,\ldots,m+1\}$ et dont la date de fin dans $\sigma_{i_{m+2}}$ est supérieure ou égale à 4. En effet, au plus m tâches de longueur 2 peuvent se terminer strictement avant la date 4 puisque m machines sont disponibles. Ainsi, la tâche d'id i_y peut, en déclarant 3 à la place de 2, diminuer sa date de fin. L'algorithme $\mathcal{A}$ n'est donc pas à véracité garantie et ceci montre qu'il n'existe pas d'algorithme déterministe à véracité garantie ayant un rapport d'approximation inférieur à $7/6$. □

6.3.2 Un algorithme déterministe avec véracité garantie

> Soient m machines et n tâches $1,\ldots,n$ qui déclarent des longueurs égales à $b_1,\ldots,b_n$.
> Soit OPT la date de fin d'un ordonnancement optimal de ces tâches.
> Soit σ_{LPT} l'ordonnancement de ces tâches obtenu avec l'algorithme de liste LPT.
> Soit $p(i)$ la machine sur laquelle la tâche i est ordonnancée dans σ_{LPT}, et C_i la date de fin de la tâche i dans σ_{LPT}.
> Retourner l'ordonnancement dans lequel, pour tout $i \in \{1,\ldots,n\}$, la tâche i est ordonnancée sur $p(i)$ et commence à la date $\left(\frac{4}{3}-\frac{1}{3\,m}\right) OPT - C_i$.

L'algorithme LPT_{miroir}

Théorème 6.8 *LPT_{miroir} est un algorithme déterministe, à véracité garantie, et $(\frac{4}{3}-\frac{1}{3m})$-approché.*

Preuve : Soient n tâches $1, \ldots, n$ ayant comme longueurs réelles $l_1, \ldots, l_n$ et comme longueurs déclarées $b_1, \ldots, b_n$. Supposons que la tâche i déclare une longueur $b_i > l_i$. De cette façon i commence plus tôt (ou à la même date) dans σ_{LPT} que si elle avait déclaré l_i. De plus, la date de fin de l'ordonnancement optimal quand i déclare $b_i > l_i$ est nécessairement supérieure ou égale à la date de fin d'un ordonnancement optimal quand i déclare l_i (dans les deux cas, les autres tâches déclarent en effet les mêmes valeurs). Soit d_i la date de début de i dans σ_{LPT}. La date de fin de la tâche i dans LPT_{miroir} est $\big(4/3 - 1/(3\,m)\big)OPT - C_i + b_i = \big(4/3 - 1/(3\,m)\big)OPT - d_i$ car $d_i = C_i - b_i$. En déclarant $b_i > l_i$, la tâche i peut seulement retarder sa date de fin dans l'ordonnancement retourné par LPT_{miroir} car OPT ne diminue pas et d_i n'augmente pas. La tâche i n'a donc pas intérêt à mentir.

Puisque le rapport d'approximation de l'ordonnancement obtenu avec l'algorithme de liste LPT est d'au plus $\big(4/3 - 1/(3\,m)\big)$ [42], l'ordonnancement retourné par LPT_{miroir} est réalisable, et son rapport d'approximation est, par construction, $\big(4/3 - 1/(3\,m)\big)$. L'algorithme LPT_{miroir} est donc un algorithme déterministe à véracité garantie $\big(4/3 - 1/(3\,m)\big)$-approché. □

Exemple. Considérons l'instance suivante : 3 machines, une tâche de longueur 5, deux tâches de longueur 3, trois tâches de longueur 2, et une tâche de longueur 1. Les numéros d'identification sont données par ordre de longueurs décroissantes. L'ordonnancement optimal de ces tâches a une date de fin OPT égale à 6 (les tâches de longueur 2 sont sur la même machine, celles de longueur 3 sur la même machine, et les autres sur la machine restante). Ainsi $(\frac{4}{3} - \frac{1}{3m})OPT = 7 + \frac{1}{3}$. La figure 6.3 *Gauche* montre σ_{LPT}, un ordonnancement LPT de ces tâches. La figure 6.3 *Droite* montre l'ordonnancement retourné par LPT_{miroir} pour ces tâches.

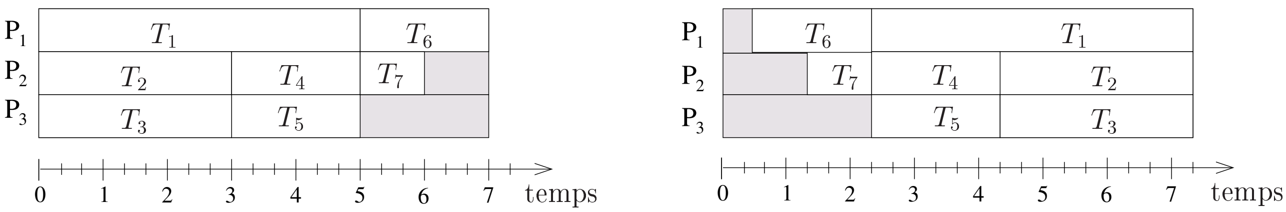


FIG. 6.3 – *Gauche :* Ordonnancement LPT. *Droite :* Ordonnancement LPT_{miroir}. T_i représente la tâche i.

Nous pouvons remarquer que LPT_{miroir} n'est pas un algorithme polynomial puisque il est nécessaire dans cet algorithme de calculer la date de fin d'un ordonnancement optimal, ce qui est un problème NP-difficile [40]. Cependant, il est possible d'obtenir un algorithme qui s'exécute en un temps polynomial et qui est $\big(4/3 - 1/(3m)\big)$-approché, même si certaines tâches ne déclarent pas leurs vraies valeurs. Cet algorithme est le suivant : nous construisons tout d'abord un ordonnancement σ_{LPT} avec l'algorithme de liste LPT. Soit $p(i)$ la machine sur laquelle se trouve la tâche i dans σ_{LPT}, soit C_i la date de fin de i dans σ_{LPT}, et soit C_{max} la date de fin de l'ordonnancement σ_{LPT}. Nous construisons alors l'ordonnancement final σ' dans lequel la tâche i est ordonnancée sur $p(i)$ et commence au temps $C_{max} - C_i$.
Cet ordonnancement est $\big(4/3 - 1/(3m)\big)$-approché, comme nous le montrerons dans le paragraphe suivant. Ceci signifie que la date de fin de l'ordonnancement retourné par cet algorithme est infé-

rieure ou égale à $\big(4/3 - 1/(3m)\big)$ multiplié par la date de fin d'un ordonnancement optimal dans lequel les tâches déclarent leurs vraies longueurs. Ainsi, bien que cet algorithme ne soit pas nécessairement à véracité garantie (nous ne savons pas si c'est le cas) son rapport d'approximation est connu et borné, ce qui est la principale raison pour laquelle l'on souhaite en général des algorithmes à garantie de véracité. Montrons maintenant que le rapport d'approximation de cet algorithme est $\big(4/3 - 1/(3m)\big)$.
Supposons que chaque tâche j ($j \in \{1, \ldots, n\}, j \neq i$) a déclaré une longueur b_j. Soit $\sigma_{LPT}(b_i)$ l'ordonnancement LPT obtenu quand i déclare b_i, soit $d_i(b_i)$ la date à laquelle la tâche i commence à être exécutée dans $\sigma_{LPT}(b_i)$, et soit $C_{max}(\sigma_{LPT}(b_i))$ la date de fin de $\sigma_{LPT}(b_i)$. La date de fin de la tâche i (qui déclare b_i) dans σ' est égale à $C_{max}(\sigma_{LPT}(b_i)) - d_i(b_i)$. Puisque avec l'algorithme de liste LPT les tâches sont ordonnancées par ordre de longueurs (déclarées) décroissantes, si $b_i > l_i$ alors $d_i(b_i) \leq d_i(l_i)$. Ainsi, quelque soient les valeurs déclarées par les autres tâches, i a intérêt à mentir et à déclarer $b_i > l_i$ seulement si $C_{max}(\sigma_{LPT}(b_i)) < C_{max}(\sigma_{LPT}(l_i))$. Puisque ceci est vrai pour chaque tâche, aucune tâche n'a intérêt à augmenter unilatéralement la valeur qu'elle déclare, à moins que cela diminue la date de fin de l'ordonnancement. La date de fin de l'ordonnancement σ' dans lequel les tâches déclarent leur vraies valeurs est $\big(4/3 - 1/(3m)\big)$-approchée, et donc la date de fin de l'ordonnancement retourné par cet algorithme sera également $\big(4/3-1/(3m)\big)$-approchée.

6.3.3 Un algorithme optimal avec véracité garantie

Nous considérons maintenant l'algorithme $RandBloc$.

• Soit un ordonnancement $\mathcal{S}$ optimal et tel que chaque tâche est ordonnancée soit au temps 0, soit immédiatement après une autre tâche. Soit C^i_{max} la date de fin de la dernière tâche sur la machine i dans $\mathcal{S}$. Soit C_{max} la date de fin de $\mathcal{S}$.
• Rajouter sur chaque machine i une tâche fictive de longueur $(C_{max} - C^i_{max})$.
• Chaque machine ordonnance ses tâches (celles qu'elle a dans $\mathcal{S}$, plus la tâche fictive) en suivant un ordre aléatoire. La tâche fictive correspondra à un temps d'inactivité sur la machine.

L'algorithme $RandBloc$

Théorème 6.9 *$RandBloc$ est un algorithme probabiliste optimal avec véracité garantie.*

Preuve : Sur chaque machine, les tâches sont ordonnancées dans un ordre aléatoire. Si x tâches $\{1, \ldots, x\}$ sont ordonnancées dans un ordre aléatoire sur une machine, alors l'espérance de la date de fin de la tâche $t \in \{1, \ldots, x\}$ est $\frac{1}{2} \sum_{i=1, i \neq t}^{x} b_i + b_t$ où b_i est la longueur de la tâche i (voir le lemme 5.1 page 87).
Sur chaque machine, la somme des longueurs des tâches est égale à la date de fin de l'ordonnancement. Ainsi, si t déclare une valeur b_t supérieure à sa longueur réelle l_t, l'espérance de sa date de fin va augmenter : elle sera de $\frac{1}{2} \sum_{i=1, i \neq t}^{x} b_i + b_t > \frac{1}{2} \sum_{i=1, i \neq t}^{x} b_i + l_t$. Cet algorithme est donc à véracité garantie. De plus, puisque la date de fin de l'ordonnancement retourné par $RandBloc$ est égale à la date de fin d'un ordonnancement optimal, l'algorithme $RandBloc$ est optimal. □

Nous pouvons remarquer que $RandBloc$ n'est pas un algorithme polynomial puisqu'il demande de calculer une solution optimale $\mathcal{S}$. Cependant, de la même façon que nous l'avons remarqué dans la section précédente, il est possible d'obtenir un algorithme polynomial ayant un rapport d'approximation égal à $(1+\varepsilon)$ et ce même si les tâches mentent. Si nous remplaçons dans l'énoncé de l'algorithme $RandBloc$ l'ordonnancement optimal $\mathcal{S}$ (obtenu en un temps exponentiel si $P \neq NP$) par un ordonnancement $(1+\varepsilon)$-approché (que l'on peut obtenir en un temps polynomial en utilisant par exemple le schéma d'approximation de Graham [42]), alors les tâches ont intérêt à mentir seulement si cela fait diminuer la date de fin de l'ordonnancement. La date de fin de l'ordonnancement sera donc inférieure ou égale à la date de fin de l'ordonnancement obtenu par l'algorithme auquel toutes les tâches déclarent leurs vraies longueurs. Cette méthode permet donc de transformer tout algorithme $(1+\varepsilon)$ approché dans le cas où il connaît la vraie longueur des tâches en un algorithme $(1+\varepsilon)$-approché dans le cas où chaque tâche peut déclarer $b_i \geq l_i$.

6.4 Mécanismes de coordination avec véracité garantie

Dans un mécanisme de coordination à véracité garantie, les tâches déclarent elles-mêmes leur longueur, et choisissent également la machine sur laquelle elles vont être exécutées. Deux modèles sont alors possibles.
Dans le premier modèle, la stratégie de chaque tâche i est un couple (M_i, b_i), où M_i est la distribution de probabilités avec laquelle la tâche i a choisi d'aller sur les différentes machines, et b_i est la longueur déclarée par i. Dans ce premier modèle, b_i et M_i sont annoncés en même temps.
Dans le deuxième modèle, la stratégie d'une tâche se fait en deux temps : les tâches déclarent d'abord leurs longueurs ; les machines ayant différentes politiques $Pol_1, \ldots, Pol_x$ (chacune avec une probabilité $p_1, \ldots, p_x$) annoncent alors parmi ces politiques la politique qu'elle effectueront réellement ; puis les tâches choisissent les machines sur lesquelles elles seront ordonnancées. Par exemple, une machine P_i peut avoir une politique Pol_1 avec une probabilité p et une politique Pol_2 avec une probabilité $1-p$. Avant de déclarer leurs longueurs, les tâches savent uniquement que la politique de P_i est Pol_1 avec une probabilité p, Pol_2 sinon. Les tâches déclarent alors leurs longueurs. Ensuite, P_i décide de sa politique (qui ne dépend pas des tâches mais est bien Pol_1 avec une probabilité p, Pol_2 sinon) et, connaissant la politique définitive de P_i, les tâches choisissent la machine sur laquelle elles seront ordonnancées.
Les résultats d'inapproximabilité que nous donnons dans la section suivante sont valables pour les deux modèles que nous venons de présenter. Le mécanisme de coordination à véracité garantie que nous présenterons dans la section 6.4.2 sera valable dans le deuxième modèle.

Nous rappelons qu'un équilibre de Nash pur est un équilibre de Nash dans lequel chaque tâche a une stratégie pure, i.e. chaque tâche choisit avec une probabilité 1 d'aller sur une machine donnée, et avec une probabilité 0 d'aller sur les autres machines. Notons que, puisque nous nous intéressons à des mécanismes de coordination qui sont avec véracité garantie, nous supposons que chaque tâche doit déclarer une seule valeur et non une distribution de valeurs, chacune avec une probabilité donnée. En effet, un mécanisme de coordination dans lequel une tâche aurait intérêt à déclarer autre chose que sa vraie longueur avec une probabilité 1 ne serait pas à véracité garantie.

6.4.1 Résultats d'inapproximabilité

Nous pouvons, à partir des résultats donnés dans la section 6.2.1, et concernant le meilleur rapport d'approximation possible pour un algorithme à véracité garantie dans le modèle fort, déduire des résultats concernant le meilleur rapport de coordination possible pour un mécanisme de coordination à véracité garantie dans le modèle fort.
Soit $\rho \geq 1$. S'il n'existe pas d'algorithme déterministe à véracité garantie ayant un rapport d'approximation de ρ, alors il n'existe pas de mécanisme de coordination déterministe à véracité garantie qui induit toujours des équilibres de Nash purs et qui a un rapport de coordination inférieur ou égal à ρ. En effet, si ce n'était pas le cas, alors l'algorithme déterministe qui consiste à construire l'ordonnancement obtenu dans un équilibre de Nash pur obtenu avec ce mécanisme de coordination ρ-approché serait un algorithme déterministe à véracité garantie ρ-approché.
De la même façon, s'il n'existe pas d'algorithme (probabiliste) à véracité garantie ayant un rapport d'approximation de ρ, alors il n'existe pas de mécanisme de coordination (probabiliste) à véracité garantie qui a un rapport de coordination inférieur ou égal à ρ. En effet, si ce n'était pas le cas, l'algorithme qui consiste à construire l'ordonnancement obtenu dans un équilibre de Nash (même mixte) induit par ce mécanisme de coordination ρ-approché et à véracité garantie serait un algorithme (probabiliste) ρ-approché à véracité garantie.

Cette observation nous mène aux résultats suivants pour le modèle fort d'exécution. Nous déduisons du théorème 6.1 qu'il n'existe pas de mécanisme de coordination déterministe à véracité garantie qui induit des équilibres de Nash purs et a un rapport de coordination inférieur à $2 - \frac{1}{m}$. Il n'existe donc pas de mécanisme de coordination déterministe induisant des équilibres de Nash purs et à véracité garantie, ayant un meilleur rapport que le mécanisme de coordination SPT, dont le rapport de coordination est de $2 - \frac{1}{m}$. Nous déduisons du théorème 6.2 qu'il n'existe pas de mécanisme de coordination à véracité garantie qui a un rapport de coordination inférieur à $\frac{3}{2} - \frac{1}{2m}$.
Nous considérons maintenant le modèle souple d'exécution. Les théorèmes 6.10 et 6.11 donnent des bornes inférieures sur le rapport de coordination qu'un mécanisme de coordination induisant des équilibres de Nash purs peut avoir dans ce modèle.

Théorème 6.10 *Il n'existe pas de mécanisme de coordination déterministe à véracité garantie qui induit des équilibres de Nash purs et qui a un rapport de coordination inférieur à* $\frac{1+\sqrt{17}}{4} \approx 1,28$.

Preuve : Prouvons tout d'abord ce résultat dans le cas où il y a deux machines, P_1 et P_2. Soit $\varepsilon > 0$. Supposons qu'il existe un mécanisme de coordination déterministe $\mathcal{M}$ qui induit des équilibres de Nash purs et qui a un rapport de coordination égal à $\frac{1+\sqrt{17}}{4} - \varepsilon$. Considérons l'instance I_1 suivante : trois tâches de longueur 1. Puisque $\mathcal{M}$ est un mécanisme de coordination déterministe qui induit des équilibres de Nash purs, il existe au moins une tâche de I_1 qui a une date de fin supérieure ou égale à 2. Soit t une telle tâche.

Considérons maintenant cette instance I_2 : deux tâches de longueur $\frac{-1+\sqrt{17}}{2} \approx 1,56$. Puisque $\mathcal{M}$ est déterministe, induit des équilibres de Nash purs et est $(\frac{1+\sqrt{17}}{4} - \varepsilon)$-approché, il y a nécessairement une tâche sur chaque machine, et chaque tâche se termine avant la date $\frac{-1+\sqrt{17}}{2} \times \frac{1+\sqrt{17}}{4} = 2$. Ainsi, quand une machine doit ordonnancer une seule tâche de longueur $\frac{-1+\sqrt{17}}{2}$, elle doit avoir terminé l'exécution de cette tâche avant la date 2.

Considérons enfin l'instance suivante I_3 : deux tâches de longueur 1, et une tâche de longueur $\frac{-1+\sqrt{17}}{2}$. Puisque $\mathcal{M}$ est déterministe, induit des équilibres de Nash purs et est $(\frac{1+\sqrt{17}}{4} - \varepsilon)$-approché, la tâche de longueur $\frac{-1+\sqrt{17}}{2}$ est nécessairement seule sur sa machine (sans perte de généralité, sur P_2). Comme nous l'avons vu dans le paragraphe précédent, P_2 doit avoir exécuté cette tâche avant la date 2. Ainsi, la tâche t de l'instance I_1 a intérêt à déclarer $\frac{-1+\sqrt{17}}{2}$ à la place de 1 : de cette manière elle se terminera avant la date 2, alors qu'elle se terminerait à une date supérieure ou égale à 2 en déclarant 1. $\mathcal{M}$ n'est donc pas un mécanisme de coordination à véracité garantie.

Il est facile d'étendre cette preuve dans le cas où il y a $m > 2$ machines, en ayant $m+1$ tâches de longueur 1 dans I_1 ; m tâches de longueur $\frac{-1+\sqrt{17}}{2}$ dans I_2 ; et m tâches de longueur 1 et une tâche de longueur $\frac{-1+\sqrt{17}}{2}$ dans I_3. □

Théorème 6.11 *Il n'existe pas de mécanisme de coordination probabiliste à véracité garantie qui induit des équilibres de Nash purs et qui a un rapport de coordination inférieur à* $1+\frac{\sqrt{13}-3}{4} \approx 1,15$.

Preuve : Nous prouvons tout d'abord ce résultat dans le cas où il y a deux machines, P_1 et P_2. Soit $\varepsilon > 0$. Supposons qu'il existe un mécanisme de coordination $\mathcal{M}$ qui induit des équilibres de Nash purs et qui a un rapport de coordination égal à $\rho < \frac{2+\alpha}{2}$ (avec $\alpha < 1$). Considérons l'instance I_1 suivante : trois tâches de longueur 1. Puisque $\mathcal{M}$ est un mécanisme de coordination qui induit des équilibres de Nash purs, il existe au moins une tâche de I_1 qui a une date de fin supérieure ou égale à $1,5$ (car il y a une machine sur laquelle il y a au moins 2 tâches). Soit t une telle tâche.

Supposons que t déclare $1+\alpha$ (avec $\alpha \leq 1$). Comme $\mathcal{M}$ induit des équilibres de Nash purs et $\rho < \frac{2+\alpha}{2}$, la tâche t est seule sur sa machine (sans perte de généralité, sur P_1), car sinon la date de fin de l'ordonnancement serait supérieure à ρ multiplié par la date de fin d'un ordonnancement optimal qui est égale à 2. Puisque $\mathcal{M}$ est à véracité garantie, l'espérance de la date de fin de t doit être supérieure ou égale à $1,5$ quand elle déclare $1+\alpha$, car sinon t aurait intérêt à mentir. Ainsi, quand la machine P_1 doit ordonnancer une seule tâche de longueur $1+\alpha$, elle ne doit pas avoir terminé l'exécution de cette tâche avant la date $1,5$.

Considérons maintenant l'instance I_2 suivante : deux tâches de longueur $1+\alpha$. Puisque $\mathcal{M}$ induit des équilibres de Nash purs et $\rho < \frac{2+\alpha}{2} < 2$, il y a une seule tâche sur chaque machine. Nous avons vu dans le paragraphe précédent que, quand il y a une tâche de longueur $1+\alpha$ sur la machine P_1, l'espérance de la date de fin de cette tâche est supérieure ou égale à $1,5$. Puisque la date de fin d'un ordonnancement optimal des tâches de I_2 est $1+\alpha$, alors le rapport de coordination de $\mathcal{M}$ doit être supérieur ou égal à $\frac{1,5}{1+\alpha}$. En posant $\frac{2+\alpha}{2} = \frac{1,5}{1+\alpha}$, nous obtenons $\alpha = \frac{\sqrt{13}-3}{2}$, et donc $\rho > 1 + \frac{\sqrt{13}-3}{4} > 1,15$.

Nous pouvons facilement étendre cette preuve dans le cas où il y a $m > 2$ machines, en ayant $m+1$ tâches de longueur 1 dans I_1 et m tâches de longueur $1+\alpha$ dans I_2. □

6.4.2 Le mécanisme de coordination $SPT \oplus_p SL$

Nous nous intéressons dans cette section à un mécanisme de coordination avec véracité garantie dans le modèle souple et dans le cas où il y a deux machines.

Considérons maintenant l'algorithme $SPT \oplus_p SL$:

Soit $p \in \mathbb{R}$ tel que $0 \leq p \leq 1$. Soient n tâches $1, \ldots, n$ à ordonnancer sur deux machines. Avec une probabilité p, retourner un ordonnancement SPT. Avec une probabilité $(1-p)$, retourner un ordonnancement SPT-LPT : un ordonnancement SPT-LPT est un ordonnancement dans lequel une machine, notée P_{SPT}, ordonnance les tâches par ordre de longueurs croissantes, et l'autre machine, notée P_{LPT}, ordonnance les tâches par ordre de longueurs décroissantes. La tâche i est ordonnancée sur P_{SPT} si la somme des longueurs des tâches plus petites que i est inférieure ou égale à la somme des longueurs des tâches plus grandes que i ; sinon i est ordonnancée sur P_{LPT}.

L'algorithme $SPT \oplus_p SL$

Nous pouvons facilement transformer l'algorithme centralisé $SPT \oplus_p SL$ en un mécanisme de coordination probabiliste. En effet, nous pouvons obtenir, comme il l'a été montré dans [30], un ordonnancement SPT-LPT en ayant une machine, P_{SPT}, qui ordonnance les tâches par ordre de longueurs croissantes, et l'autre machine, P_{LPT}, qui ordonnance les tâches par ordre de longueurs décroissantes. Ainsi une tâche i ira sur P_{SPT} si la longueur totale des tâches plus petites qu'elle est inférieure ou égale à la longueur totale des tâches plus grandes qu'elle ; sinon la tâche i choisira d'aller sur P_{LPT}. De même, comme nous l'avons montré dans l'introduction de ce chapitre, nous pouvons obtenir un ordonnancement SPT en ayant deux machines, appelées P_1 et P_2, qui ordonnancent leurs tâches par ordre de longueurs croissantes, et P_2 qui ajoute un petit temps ε avant la première tâche au tout début de l'ordonnancement. De cette façon, la plus petite tâche ira sur P_1, la deuxième plus petite tâche sur P_2, et ainsi de suite. Nous obtiendrons ainsi le seul équilibre de Nash possible, qui est un ordonnancement SPT. Le mécanisme de coordination correspondant à $SPT \oplus_p SL$ est donc le suivant :

Soient n tâches à ordonnancer sur deux machines P_1 et P_2. Soit $p \in \mathbb{R}$ tel que $0 \leq p \leq 1$. Soit $\varepsilon > 0$ une petite valeur inférieure à la longueur de chaque tâche. La machine P_1 ordonnance, à partir de la date 0, ses tâches par ordre de longueurs croissantes. La machine P_2 ordonnance avec une probabilité p ses tâches par ordre de longueurs croissantes et en commençant à exécuter la première tâche à la date ε ; et P_2 ordonnance avec une probabilité $(1-p)$ ses tâches par ordre de longueurs décroissantes, en commençant sa première tâche à la date 0.

Le mécanisme de coordination correspondant à $SPT \oplus_p SL$

Théorème 6.12 *L'espérance du rapport d'approximation de $SPT \oplus_p SL$ est $\frac{4}{3} + \frac{p}{6}$.*

Preuve : Le rapport d'approximation d'un ordonnancement SPT est $\frac{3}{2}$ (voir [42]), et le rapport d'approximation d'un ordonnancement SPT-LPT est $\frac{4}{3}$ (voir [30]). L'espérance du rapport d'approximation de $SPT \oplus_p SL$ est donc $p\,\frac{3}{2} + (1-p)\,\frac{4}{3}$, i.e. $p(\frac{3}{2} - \frac{4}{3}) + \frac{4}{3} = \frac{4}{3} + \frac{p}{6}$. □

6.4.3 Véracité garantie de $SPT \oplus_p SL$

Dans cette section, nous utilisons le modèle souple d'exécution. Quand nous supposons que les longueurs des tâches sont des puissances d'une constante C, alors nous supposons que chaque tâche peut seulement déclarer une valeur qui est une puissance de C. Si ce n'était pas le cas (i.e. si une tâche déclare une longueur qui n'est pas une puissance de C) alors nous pourrions arrondir cette valeur à la puissance de C supérieure la plus proche.

Théorème 6.13 *Soit $p \in \mathbb{R}$ tel que $\frac{2}{3} \leq p \leq 1$. L'algorithme $SPT \oplus_p SL$ est à véracité garantie si les tâches ont des longueurs qui sont des puissances d'une constante $C \geq \frac{4-3\,p}{2-p}$.*

Preuve : Soient n tâches à ordonnancer sur deux machines. Nous savons que les longueurs de ces tâches sont des puissances de C (et que donc ces tâches doivent déclarer une valeur qui est une puissance de C). Supposons qu'une tâche i, de longueur l_i, déclare $b_k > l_i$. Montrons alors que l'espérance de la date de fin de la tâche i n'est pas plus grande quand i déclare l_i plutôt que b_k. Soit $\Gamma = \{1, \ldots, i, \ldots, k, \ldots, n+1\}$ un ensemble de $n+1$ tâches (la tâche i qui déclare l_i ; les $n-1$ autres tâches à ordonnancer qui déclarent chacune une valeur ; et une tâche k qui représente la tâche i qui déclare b_k au lieu de l_i). Supposons que les longueurs déclarées par ces tâches sont $b_1 \leq \cdots \leq b_i \leq \cdots \leq b_k \leq \cdots \leq b_{n+1}$ ($\forall j \notin \{i,k\}$, b_j représente la valeur déclarée par la tâche j ; $b_i = l_i$ et $b_k > l_i$). Si la tâche i déclare l_i alors les tâches que nous devons ordonnancer sont les tâches de $\Gamma_i = \Gamma \setminus k$; si la tâche i déclare $b_k > l_i$, alors les tâches à ordonnancer sont les tâches de $\Gamma_k = \Gamma \setminus i$ (la tâche k représente la tâche i dans ce cas). L'algorithme $SPT \oplus_p SL$ est à véracité garantie si l'espérance de la date de fin de i dans l'ordonnancement des tâches de Γ_i est inférieure ou égale à la date de fin de k dans l'ordonnancement des tâches de Γ_k.

Cet algorithme est donc à véracité garantie si la *pire* espérance de la date de fin de i dans l'ordonnancement des tâches de Γ_i est toujours inférieure ou égale à la *meilleure* espérance de la date de fin de la tâche k dans l'ordonnancement des tâches de Γ_k. La plus grande date de fin de i dans un ordonnancement SPT des tâches de Γ_i est $\frac{\sum_{j=1}^{i-1} b_j}{2} + b_i$: ceci est le cas quand la tâche i commence à être exécutée quand toutes les tâches plus petites sont déjà terminées. La plus petite date de fin de k dans un ordonnancement SPT des tâches de Γ_k est $\frac{(\sum_{j=1}^{k} b_j) - b_i}{2}$: ceci est le cas quand la tâche k se termine en même temps que la tâche $k-1$.

Nous avons deux cas à examiner pour obtenir la date de fin de la tâche i dans un ordonnancement SPT-LPT de Γ_i. La tâche i est soit ordonnancée sur la machine P_{SPT} après les tâches qui ont une longueur (déclarée) inférieure à b_i, et elle se termine alors à la date $\sum_{j=1}^{i} b_j$ (ceci est le cas 1 - nous y ferons référence par la suite -), ou bien i est ordonnancée sur P_{LPT} après les tâches qui ont une longueur (déclarée) supérieure à b_i, et elle se termine alors à la date $(\sum_{j=i}^{n+1} b_j) - b_k$ (ceci est le cas 2). Le raisonnement est le même pour la date de fin de la tâche k dans un ordonnancement SPT-LPT de Γ_k : la tâche k est soit ordonnancée sur P_{SPT} et se termine à la date $(\sum_{j=1}^{k} b_j) - b_i$ (cas A), ou elle est ordonnancée sur P_{LPT} et se termine à la date $\sum_{j=k}^{n+1} b_j$ (cas B). Dans l'ordonnancement SPT-LPT de Γ_i (respectivement Γ_k), la tâche i (respectivement la tâche k) choisit entre les cas 1 et 2 (respectivement les cas A et B) celui qui minimise sa date de fin.

$SPT \oplus_p SL$ est à véracité garantie si la pire espérance de la date de fin de la tâche i dans un ordonnancement SPT de Γ_i, multipliée par p, plus l'espérance de la date de fin de la tâche i dans un ordonnancement SPT-LPT de Γ_i, multipliée par $(1-p)$, est inférieure ou égale à la meilleure espérance de la date de fin de la tâche k dans un ordonnancement SPT de Γ_k, multipliée par p, plus la date de fin de la tâche k dans un ordonnancement SPT-LPT de Γ_k, multipliée par $(1-p)$. Ainsi, $SPT \oplus_p SL$ est à véracité garantie si :

$$p\left(\frac{\sum_{j=1}^{i-1} b_j}{2} + b_i\right) + (1-p)\left(\min\left\{\begin{array}{l} \sum_{j=1}^{i} b_j \\ (\sum_{j=i}^{n+1} b_j) - b_k \end{array}\right.\right)$$

$$\leq p\left(\frac{(\sum_{j=1}^{k} b_j) - b_i}{2}\right) + (1-p)\left(\min\left\{\begin{array}{l} (\sum_{j=1}^{k} b_j) - b_i \\ \sum_{j=k}^{n+1} b_j \end{array}\right.\right)$$

$$\Leftrightarrow (1-p)\left(\min\left\{\begin{array}{l} \sum_{j=1}^{i} b_j \\ \sum_{j=i}^{n+1} b_j - b_k \end{array}\right.\right) \leq p\,\frac{(\sum_{j=i}^{k} b_j) - 3\,b_i}{2} + (1-p)\left(\min\left\{\begin{array}{l} \sum_{j=1}^{k} b_j - b_i \\ \sum_{j=k}^{n+1} b_j \end{array}\right.\right)$$

Nous avons maintenant quatre cas à étudier (les quatre combinaisons des deux cas possibles pour la tâche i et des deux cas possibles pour la tâche k) :

• *Cas 1A :* Dans l'ordonnancement SPT-LPT, la tâche i est ordonnancée sur P_{SPT}, et la tâche k est ordonnancée sur P_{SPT}.

Dans l'ordonnancement SPT, la tâche k ne se termine pas avant la tâche i car $b_k > b_i$ et les longueurs des autres tâches sont les mêmes (les tâches ordonnancées avant i dans l'ordonnancement SPT de Γ_i sont donc aussi ordonnancées avant k dans l'ordonnancement SPT de Γ_k). De même, dans l'ordonnancement SPT-LPT, la tâche k ne peut pas se terminer avant la tâche i car elles sont toutes les deux sur P_{SPT} et $b_k > b_i$. Ainsi, l'espérance de la date de fin de la tâche k n'est pas plus petite que l'espérance de la date de fin de la tâche i, et $SPT \oplus_p SL$ est donc à véracité garantie dans ce cas là.

• *Cas 2A :* Dans l'ordonnancement SPT-LPT, la tâche i est ordonnancée sur P_{LPT}, et la tâche k est ordonnancée sur P_{SPT}.

Ce cas ne peut pas se produire. En effet, si la tâche i est ordonnancée sur P_{LPT}, alors la longueur des tâches ordonnancées avant elle sur P_{LPT} ($\sum_{i+1}^{n+1} b_j - b_k$) est inférieure à la longueur des tâches qui seraient ordonnancées avant elle si elle était sur P_{SPT} ($\sum_1^{i-1} b_j$). Puisque $b_k > b_i$, la longueur des tâches ordonnancées avant la tâche k sur P_{LPT} ($\sum_{k+1}^{n+1} b_j = (\sum_{i+1}^{n+1} b_j - b_k) - \sum_{i+1}^{k}$) est inférieure à la longueur des tâches ordonnancées avant la tâche k sur P_{SPT} ($\sum_1^{k-1} b_j - b_i = \sum_1^{i-1} b_j + \sum_{i+1}^{k-1}$). La date de fin de k est donc inférieure si k va sur P_{LPT} plutôt que sur P_{SPT}, et le cas dans lequel la tâche i est ordonnancée sur P_{LPT} et la tâche k sur P_{SPT} ne se produit donc pas.

• *Cas 2B :* Dans l'ordonnancement SPT-LPT, la tâche i est ordonnancée sur P_{LPT}, et la tâche k est ordonnancée sur P_{LPT}.

$SPT \oplus_p SL$ est à véracité garantie si :

$$(1-p)\left((\sum_{j=i}^{n+1} b_j) - b_k\right) \leq p\left(\frac{(\sum_{j=i}^{k} b_j) - 3\,b_i}{2}\right) + (1-p)\sum_{j=k}^{n+1} b_j$$

$\Leftrightarrow (1-p)\ \left((\sum_{j=i}^{k} b_j) - 2\, b_k\right) \leq p\ \left(\frac{(\sum_{j=i}^{k} b_j)-3\, b_i}{2}\right)$

$\Leftrightarrow (\sum_{j=i}^{k} b_j) - 2\, b_k \leq p\ \left(\frac{(3\ \sum_{j=i}^{k} b_j)-3\, b_i - 4\, b_k}{2}\right)$

$\Leftrightarrow \frac{3\,p}{2}\, b_i \leq 2\,(1-p)\, b_k + \sum_{j=i}^{k} b_j\ \left(\frac{3\,p}{2} - 1\right)$

Il y a nécessairement une tâche entre la tâche i et la tâche k dans Γ, car sinon i et k commenceraient après les mêmes tâches - et donc en même temps - dans les ordonnancements SPT de Γ_i et de Γ_k (et de même pour les ordonnancements SPT-LPT) et, comme $b_k > b_i$, l'espérance de la date de fin de la tâche i serait inférieure à l'espérance de la date de fin de la tâche k. Nous avons donc $\sum_{j=i}^{k} b_j \geq 2\, b_i + b_k$.
Comme $p \geq \frac{2}{3}$, nous avons $(\frac{3\,p}{2} - 1) \geq 0$. $SPT \oplus_p SL$ est donc à véracité garantie si :
$\frac{3\,p}{2}\, b_i \leq 2\,(1-p)\, b_k + (2\, b_i + b_k)\ \left(\frac{3\,p}{2} - 1\right)$
$\Leftrightarrow \left(\frac{3\,p}{2} - 2\,(\frac{3\,p}{2} - 1)\right)\ b_i \leq \left(2\,(1-p) + (\frac{3\,p}{2} - 1)\right)\ b_k$
$\Leftrightarrow \left(2 - \frac{3\,p}{2}\right)\ b_i \leq \left(1 - \frac{p}{2}\right)\ b_k$
$\Leftrightarrow b_i \leq \frac{1-\frac{p}{2}}{2-\frac{3\,p}{2}}\, b_k$.
$\Leftrightarrow b_i \leq \frac{2-p}{4-3\,p}\, b_k$.

Puisque $b_k > l_i$, et les longueurs des tâches sont des puissances de $C \geq \frac{4-3\,p}{2-p}$, nous savons que $b_k \geq \frac{4-3\,p}{2-p}\, l_i$, et donc l'inégalité ci-dessus est vérifiée. Donc, si $p \geq \frac{2}{3}$, et si les longueurs des tâches sont des puissances de $C \geq \frac{4-3\,p}{2-p}$, $SPT \oplus_p SL$ est à véracité garantie.

• *Cas 1B :* Dans l'ordonnancement SPT-LPT, la tâche i est ordonnancée sur P_{SPT}, et la tâche k est ordonnancée sur P_{LPT}.

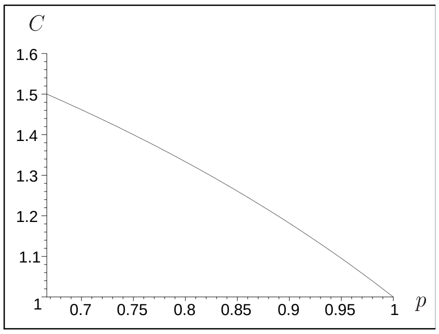
$SPT \oplus_p SL$ est à véracité garantie si :
$(1-p)\ \left(\sum_{j=1}^{i} b_j\right) \leq p\ \left(\frac{(\sum_{j=i}^{k} b_j)-3\, b_i}{2}\right) + (1-p)\ \sum_{j=k}^{n+1} b_j$

Dans l'ordonnancement SPT-LPT, la tâche i est ordonnancée sur P_{SPT} et non sur P_{LPT}, donc $\sum_{j=1}^{i} b_j \leq (\sum_{j=i}^{n+1} b_j) - b_k$. $SPT \oplus_p SL$ est donc à véracité garantie si :
$(1-p)\ \left((\sum_{j=i}^{n+1} b_j) - b_k\right) \leq p\ \left(\frac{(\sum_{j=i}^{k} b_j)-3\, b_i}{2}\right) + (1-p)\ \sum_{j=k}^{n+1} b_j$

Cette inégalité est la même que celle que nous avons rencontrée dans le cas $2B$, et donc la fin de la preuve est ici la même que la preuve du cas précédent.

Ainsi, dans tous les cas, si $p \geq \frac{2}{3}$, et si les longueurs des tâches sont des puissances d'une constante supérieure ou égale à $\frac{4-3\,p}{2-p}$, alors $SPT \oplus_p SL$ est à véracité garantie. □

La figure 6.4 *Gauche* donne une illustration du théorème 6.13 : si nous savons que les longueurs des tâches sont des puissances d'une constante supérieure ou égale à $C(p)$, alors l'algorithme $SPT \oplus_p SL$ est à véracité garantie. La figure 6.4 *Droite* illustre le théorème 6.12 et montre le rapport d'approximation en moyenne de $SPT \oplus_p SL$.



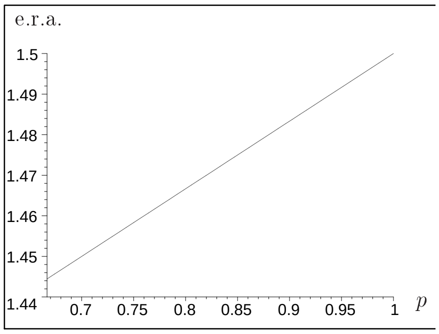


FIG. 6.4 – *Gauche :* Si les longueurs des tâches sont des puissances d'une constante supérieure ou égale à $C(p)$, alors l'algorithme $SPT\oplus_p SL$ est à véracité garantie. *Droite :* Espérance du rapport d'approximation (e.r.a) de $SPT\oplus_p SL$.

Nous avons vu que $SPT\oplus_p SL$ est à véracité garantie si les longueurs des tâches sont des puissances de $C \geq \frac{4-3p}{2-p}$. En fait, la seule condition nécessaire afin que cet algorithme soit à véracité garantie est la suivante : pour chaque tâche i, $l_{i+1} = l_i$ ou $l_{i+1} \geq C \times l_i$. Ainsi, si nous savons que les longueurs des tâches appartiennent à un ensemble $S = \{x_1, x_2, \ldots, x_k\}$ tel que pour chaque j, $x_{j+1} \geq C \times x_j$, alors $SPT\oplus_p SL$ est à véracité garantie.

Cependant, $SPT\oplus_p SL$ n'est pas à véracité garantie si les valeurs possibles des tâches ne sont pas restreintes, comme le montre le théorème 6.14, et n'est pas à véracité garantie si $p < \frac{1}{2}$, et ceci même si les tâches sont des puissances de n'importe quel nombre entier $B > 1$ (voir théorème 6.15).

Théorème 6.14 *Soit $p \in \mathbb{R}$ tel que $0 \leq p < 1$. L'algorithme $SPT\oplus_p SL$ n'est pas à véracité garantie si les longueurs des tâches peuvent être quelconques.*

Preuve : Soit ε tel que $0 \leq \varepsilon < 1 - p$. Considérons les tâches suivantes : deux tâches 1 et 2 de longueur 1, une tâche 3 de longueur $2-\varepsilon$ et une tâche 4 de longueur 2. Si la tâche 3 déclare sa vraie valeur, $2 - \varepsilon$, son espérace de date de fin est $p\,(3-\varepsilon) + (1-p)\,(4-\varepsilon)$, car elle est ordonnancée après une tâche de longueur 1 dans l'ordonnancemennt SPT, et après la tâche de longueur 2 dans l'odonnancement SPT-LPT. Si la tâche 3 ne déclare pas sa vraie longueur mais déclare $2+\varepsilon$ à la place, l'espérance de sa date de fin est alors $p\,(3+\varepsilon)+(1-p)\,(2+\varepsilon)$. En effet, dans ce cas la tâche 3 est ordonnancée dans l'ordonnancement SPT après une tâche de longueur 1, et est ordonnancée en première position sur P_{LPT} dans l'ordonnancement SPT-LPT.

Ainsi, quelque soit la valeur de p, l'espérance de la date de fin de la tâche 3 est plus petite quand cette tâche déclare $2+\varepsilon$ plutôt que sa vraie longueur $2-\varepsilon$. Cet algorithme n'est donc pas à véracité garantie si les longueurs possibles des tâches ne sont pas restreintes. □

Théorème 6.15 *Soit $p \in \mathbb{R}$ tel que $0 \leq p < \frac{1}{2}$. L'algorithme $SPT\oplus_p SL$ n'est pas à véracité garantie, même si les longueurs des tâches sont des puissances d'un entier B ($B > 1$), quelque soit la valeur de B.*

Preuve : Soit $B \in \mathbb{N}$ tel que $B > 1$, et $p \in \mathbb{R}$ tel que $p < \frac{1}{2}$. Supposons que les longueurs des tâches sont des puissances de B (et donc déclarent une valeur qui est une puissance de B). Nous montrons que $SPT \oplus_p SL$ n'est pas à véracité garantie en montrant qu'il existe une instance dans laquelle une tâche t peut diminuer l'espérance de sa date de fin en déclarant une longueur supérieure à sa vraie longueur. Soit $x \in \mathbb{N}$ un nombre pair supérieur à $\frac{B^2-1}{(B+p\,(\frac{1}{2}-2\,B))}$. Considérons l'instance suivante : $(x\,B+1)$ tâches de longueurs B, et x tâches de longueur B^2. Soit t la dernière tâche de longueur B (i.e. la tâche de longueur B qui a le plus grand numéro d'identification : $t = x\,B+1$ si les tâches $\{1, 2, \ldots, x\,B+1\}$ sont celles de longueur B). Montrons que l'espérance de la date de fin de t dans $SPT \oplus_p SL$ est inférieure si t déclare B^3 plutôt que B.

La date de fin de t qui déclare B est $\frac{x\,B}{2} + B$ dans l'ordonnancement SPT, et $x\,B^2 + B$ dans l'ordonnancement SPT-LPT (voir figure 6.5). L'espérance de la date de fin de t déclarant B dans $SPT \oplus_p SL$ est donc $p\,(\frac{x\,B}{2} + B) + (1-p)\,(x\,B^2 + B) = p(\frac{x\,B}{2} - x\,B^2) + (x\,B^2 + B)$.

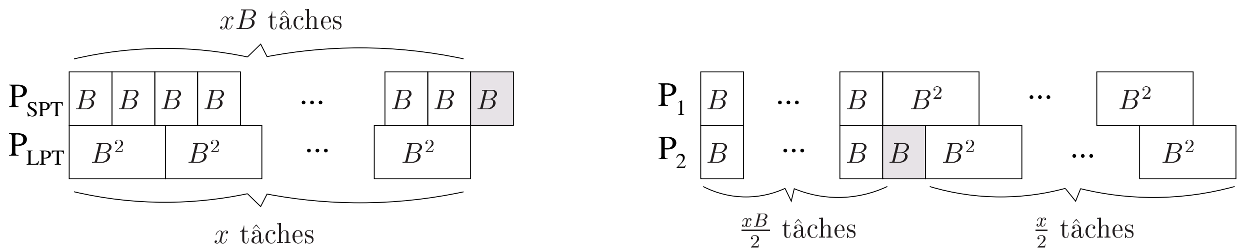


FIG. 6.5 – Ordonnancements retournés quand la tâche t, colorée en gris, déclare B. *Gauche :* Ordonnancement SPT-LPT, *Droite :* ordonnancement SPT. Le nombre à l'intérieur de chaque tâche représente sa longueur.

La date de fin de t qui déclare B^3 est $x\,B^2 + B^3$ dans l'ordonnancement SPT (t est en dernière position sur cet ordonnancement, après $\frac{x\,B}{2}$ tâches de longueur B et $\frac{x}{2}$ tâches de longueur B^2), et la date de fin de t qui déclare B^3 est B^3 dans l'ordonnancement SPT-LPT (t en première position sur P_{LPT}). L'espérance de la date de fin de t dans $SPT \oplus_p SL$ est donc $p(x\,B^2 + B^3) + (1-p)(B^3) = pxB^2 + B^3$.

$SPT \oplus_p SL$ n'est pas à véracité garantie si l'espérance de la date de fin de t qui déclare B^3 est inférieure à l'espérance de la date de fin de t qui déclare B. Il n'est donc pas à véracité garantie si :
$p\,x\,B^2 + B^3 < p\,(\frac{x\,B}{2} - x\,B^2) + (x\,B^2 + B)$
$\Leftrightarrow B^3 - B < -p\,x\,B^2 + \frac{p\,x\,B}{2} - p\,x\,B^2 + x\,B^2$
$\Leftrightarrow B^3 - B < x\,(B^2 + p\,(\frac{B}{2} - 2\,B^2))$
$\Leftrightarrow B^2 - 1 < x\,(B + p\,(\frac{1}{2} - 2\,B))$
$x > \frac{B^2-1}{(B+p\,(\frac{1}{2}-2\,B))}$, car $B + p\,(\frac{1}{2} - 2\,B) > 0$ puisque $0 \leq p < \frac{1}{2}$.
Puisque x est un nombre pair supérieur à $\frac{B^2-1}{(B+p\,(\frac{1}{2}-2\,B))}$, x respecte la condition ci-dessus et donc $SPT \oplus_p SL$ n'est pas à véracité garantie si $p < \frac{1}{2}$. □

6.4.4 Autres mécanismes de coordination : résultats négatifs

Nous considérons maintenant les autres mécanismes de coordination basés sur les combinaisons d'ordonnancements SPT, LPT et SPT-LPT.

$SPT \oplus_p LPT$ est l'algorithme qui retourne avec une probabilité p un ordonnancement SPT, et avec une probabilité $(1-p)$ un ordonnancement LPT. $LPT \oplus_p SL$ est l'algorithme qui retourne avec une probabilité p un ordonnancement LPT, et avec une probabilité $(1-p)$ un ordonnancement SPT-LPT. Nous avons vu dans la section 6.4.2 qu'il existe un mécanisme de coordination qui retourne un ordonnancement SPT ou un ordonnancement SPT-LPT. De même, en ajoutant des petits délais entre les tâches sur chaque machine – qui ordonnance les tâches de la plus petite à la plus grande –, les auteurs ont donné dans [30] un mécanisme de coordination qui retourne un ordonnancement LPT (les délais entre les tâches étant négligeables). Le mécanisme de coordination correspondant à $SPT \oplus_p LPT$ (respectivement $LPT \oplus_p SL$) est donc le mécanisme dans lequel chaque machine a avec une probabilité p la politique qu'elle aurait eue dans le mécanisme de coordination correspondant à SPT (respectivement LPT), et avec une probabilité $(1-p)$ la politique qu'elle aurait eue dans le mécanisme de coordination correspondant à SPT-LPT (respectivement LPT). Notons cependant que, contrairement au mécanisme de coordination $SPT \oplus_p SL$ et $LPT \oplus_p SL$, il est nécessaire pour le mécanisme de coordination corespondant à $SPT \oplus_p LPT$ d'avoir une certaine coordination entre les machines puisque les machines doivent toutes avoir en même temps la politique SPT ou toutes la politique LPT. La probabilité de choisir entre la politique SPT ou la politique LPT ne doit donc pas être décidée indépendamment par chaque machine.

Nous montrons maintenant que $SPT \oplus_p LPT$ n'est pas à véracité garantie si les longueurs des tâches ne sont pas restreintes (voir théorème 6.16) ; et que, si $p < \frac{1}{2}$, $SPT \oplus_p LPT$ n'est pas à véracité garantie même si les longueurs des tâches sont des puissances d'une certaine constante B, et quelque soit la valeur de B (voir théorème 6.17). Nous montrons également que $LPT \oplus_p SL$ n'est pas à véracité garantie, quelque soit p, et même si les longueurs des tâches sont des puissances d'une constante B, quelque soit la valeur de B (voir théorème 6.18).

Théorème 6.16 *Soit $p \in \mathbb{R}$ tel que $0 \leq p < 1$. L'algorithme $SPT \oplus_p LPT$ n'est pas à véracité garantie si les longueurs des tâches peuvent être quelconques.*

Preuve : Soit ε tel que $0 \leq \varepsilon < \frac{1-p}{1+p}$. Considérons l'instance suivante : une tâche t de longueur 1, une tâche de longueur $1+\varepsilon$ et une tâche de longueur 2. Si la tâche t déclare sa vraie longueur, 1, son espérance de la date de fin est $f_{vrai} = p + (1-p)(2+\varepsilon)$, car t est ordonnancée au début de l'ordonnancement SPT, et après la tâche de longueur 1 dans l'ordonnancement LPT. Si t ne déclare pas sa vraie longueur mais déclare $1 + 2\,\varepsilon$ à la place, son espérance de la date de fin est $f_{faux} = (1+2\varepsilon)$, car elle est ordonnancée en première position dans les ordonnancements SPT et LPT. Comme $\varepsilon < \frac{1-p}{1+p}$, f_{faux} est inférieure à f_{vrai}, et donc $SPT \oplus_p LPT$ n'est pas à véracité garantie. □

Théorème 6.17 *Soit $p \in \mathbb{R}$ tel que $0 \leq p < \frac{1}{2}$. L'algorithme $SPT \oplus_p LPT$ n'est pas à véracité garantie, même si les longueurs des tâches sont des puissances d'une constante B ($B > 1$), quelque soit la valeur de B.*

Preuve : Soit B et p deux nombres réels tels que $B > 1$ et $0 \leq p < \frac{1}{2}$. Supposons que les tâches ont des longueurs qui sont des puissances de B (et qu'elles doivent donc déclarer une longueur qui est une puissance de B). Nous montrons que $SPT{\oplus_p}LPT$ n'est pas à véracité garantie en montrant qu'il existe une instance dans laquelle une tâche t peut diminuer l'espérance de sa date de fin en mentant.

Soit $x = \lfloor \frac{B^2-1}{B\,(1-2p)} \rfloor + 1$. Considérons l'instance suivante : $2\,x$ tâches de longueur B^2, et une tâche, t, de longueur B. Montrons que l'espérance de date de fin de t dans $SPT{\oplus_p}LPT$ est plus petite quand elle déclare B^3 plutôt que B.
La date de fin de t qui déclare B est B dans l'ordonnancement SPT, et $x\,B^2+B$ dans l'ordonnancement LPT dans lequel t est ordonnancée après x tâches de longueur B^2. L'espérance de la date de fin de t déclarant B dans $SPT{\oplus_p}LPT$ est alors $f_{vrai} = p\,B+(1-p)\,(x\,B^2+B) = B+(1-p)\,x\,B^2$.
La date de fin de t qui déclare B^3 est $x\,B^2 + B^3$ dans l'ordonnancement SPT (elle est ordonnancée après x tâches de longueur B^2), et B^3 dans l'ordonnancement LPT. L'espérance de la date de fin de t déclarant B^3 dans $SPT{\oplus_p}LPT$ est alors $f_{faux} = p\,(x\,B^2 + B^3) + (1-p)\,(B^3) = B^3 + p\,x\,B^2$.

$SPT{\oplus_p}LPT$ n'est pas à véracité garantie si $f_{vrai} < f_{faux}$:
$f_{vrai} < f_{faux}$
$\Leftrightarrow B^3 + p\,x\,B^2 < B + (1-p)\,x\,B^2$
$\Leftrightarrow B^3 - B < x\,B^2\,(1-2p)$
$\Leftrightarrow x > \frac{B^2-1}{B\,(1-2p)}$
Comme $x = \lfloor \frac{B^2-1}{B\,(1-2p)} \rfloor + 1$ respecte la condition ci-dessus, l'espérance de la date de fin de t dans notre instance est inférieure quand t déclare B^3 plutôt que sa vraie valeur B, et donc $SPT{\oplus_p}LPT$ n'est pas à véracité garantie. □

Théorème 6.18 *Soit $p \in \mathbb{R}$ tel que $0 \leq p \leq 1$. L'algorithme $LPT{\oplus_p}SL$ n'est pas à véracité garantie, même si les longueurs des tâches sont des puissances d'une constante B ($B > 1$), quelque soit la valeur de B.*

Preuve : Soit B et p deux nombres réels tels que $B > 1$ et $0 \leq p \leq 1$. Comme dans la preuve du théorème précédent, nous supposons que les tâches ont des longueurs qui sont des puissances de B (et qu'elles doivent donc déclarer une longueur qui est une puissance de B), et nous montrons que $SPT{\oplus_p}LPT$ n'est pas à véracité garantie en donnant une instance dans laquelle une tâche t peut diminuer l'espérance de sa date de fin en mentant. Considérons l'instance suivante : $2\,B+1$ tâches de longueur B^2, et $2\,B^2$ tâches de longueur B. Soit t la dernière tâche de longueur B^2 (i.e. la tâche de longueur B^2 qui a le plus grand numéro d'identification).
La date de fin de t qui déclare B^2 est $B^3 + B^2$ dans un ordonnancement LPT (t est ordonnancée après B tâches de longueur B^2) et $2\,B^3 + B^2$ dans l'ordonnancement SPT-LPT (t est ordonnancée après $2\,B^2$ tâches de longueur B). La date de fin de t qui déclare B^3 est B^3 dans l'ordonnancement LPT, et B^3 dans l'ordonnancement SPT-LPT (t est ordonnancée au début des deux ordonnancements). L'espérance de la date de fin de t est inférieure quand elle déclare B^3 plutôt que B^2, et donc $LPT{\oplus_p}SL$ n'est pas à véracité garantie. □

Dans les résultats négatifs de cette section, nous avons utilisé le modèle souple d'exécution : nous avons supposé que si une tâche i déclare une valeur $b > l_i$ alors sa durée d'exécution sera

b. Bien sûr, ces résultats sont aussi valables dans le modèle fort d'exécution, dans lequel le temps d'exécution d'une tâche i qui déclare $b > l_i$ reste l_i (la tâche i n'a pas à attendre b unités de temps après son départ avant d'obtenir son résultat).

6.5 Conclusion

Nous avons montré que, dans le modèle d'exécution fort, l'algorithme de liste SPT, qui a un rapport d'approximation égal à $2 - 1/m$ est le meilleur algorithme déterministe à véracité garantie. Nous avons également donné un algorithme probabiliste à véracité garantie ayant un rapport meilleur que celui de SPT (par exemple si $m = 2$ son rapport est d'environ $1,39$ alors que celui de SPT est de $1,5$), et nous avons montré qu'il n'existe pas d'algorithme (probabiliste) à véracité garantie ayant un rapport d'approximation inférieur à $3/2 - 1/(2\,m)$. Au contraire, si nous relâchons les contraintes sur le modèle d'exécution, i.e. si le résultat d'une tâche qui déclare b est retourné à cette tâche seulement b unités de temps après le début de son exécution, alors nous pouvons obtenir de meilleurs résultats. Dans ce modèle d'exécution, il existe notamment un algorithme déterministe à véracité garantie $4/3 - 1/(3\,m)$-approché, et un algorithme probabiliste optimal à véracité garantie. De plus, pour chacun de ces modèles, nous avons également donné des bornes sur le rapport de coordination qu'un mécanisme de coordination à véracité garantie peut avoir.

Une perspective immédiate serait d'améliorer les résultats dont les bornes inférieures et supérieures ne coïncident pas. Il serait notamment intéressant de savoir s'il existe un algorithme à véracité garantie meilleur que $LPT \oplus DSPT$ pour le modèle fort, ainsi que d'améliorer les bornes dans le cas des mécanismes de coordination. Par ailleurs, nous pensons qu'il n'existe pas, dans le modèle souple, d'algorithme déterministe meilleur que LPT_{miroir}.

Nous avons considéré dans ce chapitre que les tâches ne peuvent pas se diviser, et ne déclarent donc pas de longueur inférieure à leur longueur réelle. Ceci semble une hypothèse raisonnable car dans de nombreuses applications (réservations de ressources, programmes informatiques, etc.) les tâches ne peuvent pas être divisées et exécutées indépendamment à des moments et sur des machines différentes. Cependant, si l'on considère qu'il est possible pour une tâche de se diviser en plusieurs parties, chacune d'elle formant une tâche petite tâche, et si l'on considère qu'une tâche t se divisera ainsi en plusieurs petites tâches si la date de fin de la dernière de ses petites tâches est inférieure à la date de fin de t si elle déclare sa vraie longueur, alors il ne semble pas possible d'avoir un algorithme à véracité garantie sans recourir à des paiements qui inciteraient les tâches à déclarer leur vraies longueurs.

Enfin, nous avons supposé qu'un agent ne possède qu'une seule tâche. Nous pourrions également considérer le cas dans lequel il est possible pour un agent d'introduire de fausses tâches dans le système afin que la vraie tâche de l'agent soit exécutée plus rapidement dans l'ordonnancement (retourné par l'algorithme) dans lequel se trouvent ces fausses tâches, plutôt que dans l'ordonnancement sans ces nouvelles tâches. Les algorithmes déterministes SPT et LPT_{miroir} sont également à véracité garantie dans ce cas là. Les algorithmes probabilistes que nous avons donnés ne sont par contre pas à véracité garantie dans ce contexte. L'algorithme $LPT \oplus DSPT$ n'est en effet pas à véracité garantie car un agent i aurait intérêt à introduire deux tâches – la tâche t_i avec sa vraie longueur et une très grande tâche t'_i (commençant comme t_i) –, ce qui lui permettrait de diminuer la date de fin de sa tâche (sa tâche commençant à la plus petite date entre le début de t_i et le début de t'_i, selon que l'ordonnancement retourné soit un ordonnancement DSPT ou un ordonnancement LPT). L'algorithme $RandBloc$ n'est également pas à véracité garantie car la réplication de la vraie tâche de nombreuses fois lui permettrait d'être exécutée plus rapidement.

Chapitre 7

Routage de tâches autonomes dans les chemins, les arbres, et les anneaux

Dans ce chapitre, nous nous intéressons au routage de paquets autonomes - et dont le but est d'arriver le plus tôt à leur destination - dans des réseaux, et nous étudions la perte de performance due au fait de ne pas avoir d'algorithme centralisé imposant un routage aux tâches. Ces travaux ont été présentés dans [11].

7.1 Introduction

De nombreux réseaux, dont internet, utilisent un routage store-and-forward. Le problème de routage de paquets dans un réseau store-and-forward est le suivant. Nous avons un réseau représenté par un graphe orienté dans lequel les sommets sont les commutateurs (switches) et les arcs sont les liens de communication. Nous avons également un ensemble de paquets, chacun étant caractérisé par une longueur (par exemple la taille du paquet en octets), et par un couple (source, destination). Chaque paquet doit être routé de sa source vers sa destination. En général, un sommet peut être la source ou la destination d'un nombre quelconque de paquets. Nous étudierons dans ce chapitre le problème dans lequel tous les paquets partent d'une même source. La caractéristique d'un réseau store-and-forward (littéralement "garder-et-faire suivre") est la suivante : au plus un paquet peut être routé à un instant donné sur un lien donné, et un paquet ne peut pas être routé sur plusieurs liens en même temps. Le temps dont a besoin un paquet pour traverser un lien est proportionnel à sa longueur (nous supposons par la suite, sans perte de généralité, que ce temps est égal à la longueur du paquet). Avant le début du routage (qui commence à la date 0), tous les paquets sont stockés à la source dans des buffers (aussi appelés tampons). Pendant le routage, il est possible que des paquets doivent être stockés dans des buffers à des sommets intermédiaires, en attendant qu'un lien qui est déjà en train de router un paquet se libère.

Nous nous intéressons à des paquets détenus par des agents (utilisateurs) autonomes et individualistes. Chaque paquet (ou son agent) doit décider à chaque sommet, de la source à la destination, le lien qu'il emprunte. Son but est de minimiser sa date d'arrivée à sa destination, et, en fonction des stratégies des autres paquets (i.e. des liens choisis par les autres tâches) et des politiques de routage des liens (l'ordre dans lequel les liens choisissent de router les tâches qui attendent dans un buffer d'être routées sur ce lien), chaque paquet choisit un lien qui minimise sa date d'arrivée.

Notre but est de mesurer l'impact de l'absence de contrôle global (un algorithme centralisé qui affecterait toutes les tâches aux liens) sur la qualité des solutions, par rapport à une fonction objectif globale, qui est une mesure d'efficacité globale du routage. Nous considérons par la suite que la taille maximum des buffers n'est pas un paramètre du problème, et nous mesurons la qualité des solutions pour les deux fonctions objectifs suivantes (que nous souhaitons minimiser) : la date à laquelle toutes les tâches sont arrivées à leur destination, et la date moyenne à laquelle les tâches sont arrivées à leurs destinations.

Topologie du réseau. Nous étudions trois topologies de réseaux : les chemins, les arbres, et les anneaux. Pour chacune de ces topologies nous considérons deux cas : le cas où les paquets ont la même source mais des destinations différentes, et le cas où les paquets ont la même source et la même destination. Si le réseau est un chemin ou un arbre, alors il existe un chemin unique entre la source et chaque destination, et chaque paquet choisit donc à chaque étape le prochain lien vers sa destination. Au contraire, si le réseau est un anneau, alors les paquets doivent choisir à la source un des deux liens sortants de la source. Dans ce cas là, nous supposons que les paquets (ou agents) sont rationnels et nous nous intéressons à la qualité des solutions qui sont des équilibres de Nash purs, i.e. chaque paquet a une stratégie pure (il choisit un seul lien sortant d'un sommet) et aucun paquet n'a intérêt à changer unilatéralement de stratégie (étant données les stratégies des autres paquets, chaque paquet ne diminuera pas sa date d'arrivée en changeant de stratégie).

Politiques locales. Nous considérons que nous sommes dans un contexte complètement distribué. Nous avons vu que les agents se placent eux-mêmes sur le lien qu'ils ont choisi (ou, plus précisément, dans le buffer d'attente de ce lien, les tâches ayant demandées à être routées sur un lien attendant dans le buffer correspondant à ce lien que le lien les route). À chaque instant, chaque lien ne connaît que les tâches qui ont demandé à être routées sur lui. Les liens appliquent une *politique locale* commune qui leur permet de donner une priorité aux paquets : quand un lien est inactif (i.e. quand il n'est pas en train de router une tâche), il choisit, selon ce que lui dicte sa politique, un paquet à router parmi les paquets, s'il y en a, dans son buffer. La politique des liens est décidée une fois pour toutes avant le début du routage et est indépendante des instances du problème de routage. Elle est également publique, i.e. connue des agents.

Nous étudions les performances obtenues avec les politiques suivantes :

- SPT (pour *Shortest Processing Time*) : le paquet qui a la longueur la plus petite est ordonnancé en premier.
- LPT (pour *Largest Processing Time*) : le paquet qui a la plus grande longueur est ordonnancé en premier.
- LRD (pour *Largest Remaining Distance*) : le paquet dont la distance restante pour atteindre sa destination est la plus grande est ordonnancé en premier. La distance restante d'un paquet pour aller à sa destination est égale au nombre d'arcs que le paquet doit traverser pour arriver à sa destination.
- LRT (pour *Largest Remaining Time*) : le paquet dont le temps nécessaire pour arriver à sa destination est le plus grand est ordonnancé en premier. Le temps nécessaire à un paquet pour arriver à sa destination est égal à la longueur du paquet, multipliée par le nombre d'arcs que le paquet doit traverser pour arriver à sa destination.

Le prix de l'anarchie. Notre but dans ce chapitre est d'étudier la perte de performance causée par l'absence de contrôle centralisé pour le problème de routage dans des réseaux store-and-forward, et plus précisément, la qualité de la solution obtenue en utilisant les politiques locales de routage énoncées ci-dessus. Pour cela, nous étudions le rapport, dans le pire des cas (parmi toutes les instances possibles), entre la valeur de la fonction objectif globale dans la solution obtenue avec les politiques locales, et la valeur de la fonction objectif globale dans une solution optimale. Nous rappelons que les deux fonctions objectifs globales étudiées sont la date maximale d'arrivée des paquets, et la date moyenne d'arrivée des paquets. Dans le cas des anneaux, comme nous nous intéressons aux solutions qui sont des équilibres de Nash, ce rapport correspond donc à la valeur de la fonction objectif dans le pire équilibre de Nash, par rapport à la valeur de la fonction objectif dans une solution optimale. Ce rapport est donc égal au *prix de l'anarchie* obtenu quand cette politique est utilisée (le prix de l'anarchie est introduit dans la section 1.2.3). Par la suite, nous utiliserons aussi l'expression "le prix de l'anarchie d'une politique" qui désigne le prix de l'anarchie quand tous les liens du réseau utilisent cette politique. Si le réseau est un chemin ou un arbre, les agents n'ont pas vraiment la possibilité de choisir leur chemin jusqu'à leur destination (ce chemin est unique), et nous utilisons donc la notion de *rapport d'approximation* des politiques locales : le rapport d'approximation d'une politique Pol est égal au rapport maximal (parmi toutes les instances) entre la valeur de la fonction objectif globale obtenue quand la politique des liens est Pol et que chaque tâche demande à être routée sur le prochain lien vers sa destination, et la valeur de la fonction objectif globale dans une solution optimale. Notons que ce rapport d'approximation peut aussi être vu comme le prix de l'anarchie, puisque dans ce cas nous sommes dans un jeu qui a un équilibre de Nash unique (et trivial).

Exemples. La figure 7.1 illustre le résultat du routage avec les différentes politiques pour l'instance suivante : l'arbre représenté sur la figure 7.1, dans lequel le sommet S est la source. Les arcs sont orientés de la source vers les autres sommets, ou bien il y a entre deux sommets u et v voisins un arc (u, v) et un arc (v, u) (le résultat du routage sera le même, chaque tâche ayant de toutes façons intérêt à demander à être routée vers le prochain sommet vers sa destination). Il y a trois tâches : la tâche 1 a une longueur égale à 1, la tâche 2 a une longueur égale à 2, et la tâche 3 a une longueur égale à 4. Le sommet d_i ($i \in \{1, 2, 3\}$) représente la destination de la tâche i. Avec la politique SPT, les tâches partent de la source dans l'ordre suivant : 1,2,3 ; avec la politique LPT dans l'ordre inverse (3,2,1) ; avec la politique LRD dans l'ordre 1,3,2 ; et avec la politique LRT dans l'ordre 3,1,2. La date d'arrivée de chaque tâche est indiquée à côté de sa destination. Pour cette instance et pour le problème de la Date de Fin Maximum, la politique LRT est la meilleure (la date de fin maximum est de 8 avec la politique LRT, contre 9 avec la politique LRD, 10 avec la politique LPT, et 11 avec la politique SPT).

La figure 7.2 illustre une instance de routage dans un anneau. Il y a deux tâches : la tâche 1 a une longueur égale à 3 et a pour destination le sommet d_1, et la tâche 2 a une longueur égale à 1 et a pour destination le sommet d_2. Le sommet S est la source. Nous considérons que la politique des liens est LPT. Le seul équilibre de Nash ici est la solution dans laquelle la tâche 1 emprunte le côté gauche de l'anneau (les deux arcs qui la séparent de sa destination) et arrive à sa destination au temps 6, et la tâche 2 emprunte le côté droit de l'anneau et arrive à sa destination également au temps 6. Une solution dans laquelle les deux tâches emprunteraient le côté gauche de l'anneau ne serait pas un équilibre de Nash car la tâche 2 finirait alors au temps 8, et elle pourrait diminuer sa

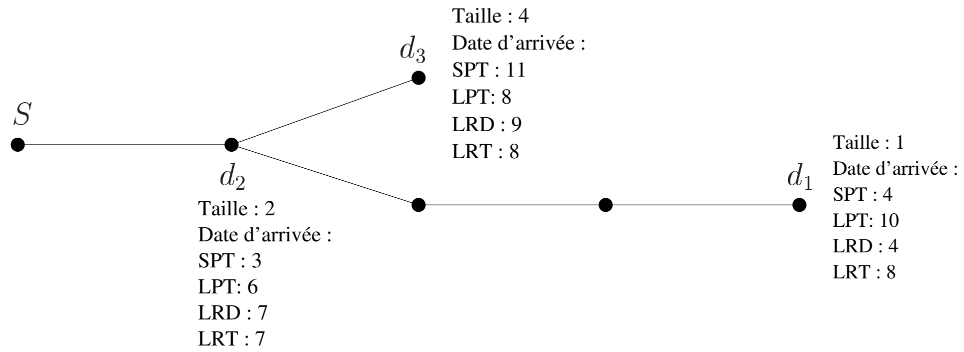


FIG. 7.1 – Exemple illustrant les différentes politiques de routages sur une instance.

date d'arrivée en passant par le côté droit de l'anneau.

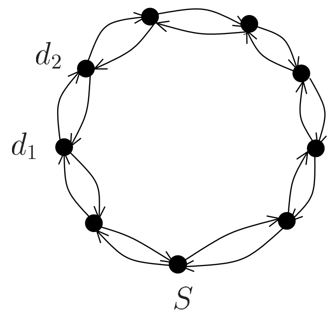


FIG. 7.2 – Une instance du problème de routage dans un anneau.

7.1.1 Travaux connexes

Le problème de *routage de paquets dans des réseaux store-and-forward* généraux est un problème NP-difficile [75] qui a été l'objet de nombreux travaux (voir par exemple [58, 59, 19, 66]). Cependant, dans ces travaux, les auteurs considèrent que tous les paquets ont *la même longueur*, et les auteurs donnent principalement des algorithmes centralisés (approchés) pour résoudre ce problème. Des algorithmes distribués ou dynamiques (online) ont aussi été proposés dans [58, 19, 66], mais, dans ces algorithmes, la politique de chaque lien ne dépend pas uniquement des paquets qui empruntent ce lien car elle dépend du nombre total de paquets dans le réseau.

Nos travaux sont également à rapprocher des problèmes de *flots qui évoluent au cours du temps* ("multicommodity flow over time problem" en anglais). Dans ces problèmes, un flot ne voyage pas instantanément à travers un réseau (un graphe) mais met un certain temps à voyager, sachant que chaque arc a un débit maximal. Dans [47], différents flux, ayant une source et une destination, doivent se déplacer de leur source vers leur destination dans un tel réseau. Les auteurs souhaitent minimiser la date à laquelle tous les flux sont arrivés à leur destination (i.e. ils souhaitent minimiser la date à laquelle le dernier flux arrive à sa destination). Une solution de ce problème est, pour chaque instant dans le temps, un débit (venant d'un ou plusieurs flux) entrant dans chaque arc. Les auteurs montrent que ce problème est NP-difficile, et proposent un algorithme centralisé approché pour le résoudre. Ils donnent aussi un algorithme glouton dans le cas où le degré sortant de chaque sommet est d'au plus un. Nous utiliserons ce résultat dans la section 7.2.3.

Mais les travaux les plus proches de ce chapitre sont peut-être ceux de G. Christodoulou, E. Koutsoupias et A. Nanavati qui ont travaillé sur des *mécanismes de coordination* [30] (voir section 1.2.4) pour un problème de routage sur m liens parallèles : n tâches partent d'une même source et désirent aller à une même destination, à travers un simple réseau constitué de m liens parallèles qui connectent la source à la destination. Chaque lien a une politique locale qui détermine l'ordre dans lequel les tâches allouées à ce lien seront routées. Chaque tâche est caractérisée par une longueur et souhaite adopter une stratégie - le choix du lien où elle sera routée - qui minimise sa date d'arrivée à sa destination. La fonction objectif globale (que l'on souhaite minimiser) est la date à laquelle tous les paquets sont arrivés à leur destination (i.e. la date d'arrivée maximale d'un paquet à sa destination). Si la politique des liens est LPT alors le meilleur mécanisme de coordination est obtenu quand la politique des liens est LPT et a un rapport de $4/3 - 1/(3\,m)$ si l'on s'intéresse aux équilibres de Nash purs et un rapport qui tend vers $4/3 - 1/(3\,m)$ si l'on s'intéresse aux équilibres de Nash mixtes (dans ce cas là des délais négligeables sont placés entre les tâches pour les forcer à avoir un équilibre de Nash pur). Récemment, dans [51], les auteurs étudient ce même problème dans le cas où les liens (ou machines) ont des vitesses différentes. Ils comparent le prix de l'anarchie avec plusieurs politiques : SPT, LPT, et Randomized (cette politique ordonnance les tâches dans un ordre aléatoire).

Il est aussi utile ici de mentionner deux autres modèles qui considèrent le problème de routage de paquets "individualistes" dans des réseaux. Tout d'abord, le modèle introduit par T. Roughgarden et E. Tardos dans [72] consiste à router dans un réseau un très grand nombre de paquets de longueurs négligeables, et mène à un problème de flot dans lequel chaque paquet veut minimiser sa date d'arrivée et l'objectif global est de minimiser la date d'arrivée du dernier paquet. Le deuxième modèle considère les *jeux de congestion* ("congestion games" en anglais), introduits par R. Rosenthal dans [71] et étudiés notamment dans [29, 24]. Dans un jeu de congestion, chaque paquet veut aller d'une source à une destination (des sommets d'un graphe) et minimiser son coût de parcours. Le coût de parcours d'un paquet i est égal à la somme des coûts des liens traversés par i, et le coût d'un lien est égal au nombre de paquets qui utilisent ce lien (ou est égal à une fonction du nombre de paquets empruntant ce lien). Les auteurs de [24] considèrent une version pondérée de ce jeu, dans laquelle les tâches ont des longueurs, et le coût d'un lien est une fonction de la somme des longueurs des liens empruntant ce lien. Dans les deux cas, la fonction objectif global consiste à minimiser le coût de parcours maximum. Ce modèle présente des différences importantes avec le notre. Ce modèle ne représente pas les réseaux store-and-forward : par exemple, deux tâches de longueurs différentes empruntant le même chemin auront le même coût. Le coût d'un lien ne prend pas en compte la date à laquelle les paquets empruntent effectivement le lien. Ainsi, un lien utilisé par deux tâches ajoutera un coût de 2 à chacune de ces tâches, même si aucune tâche ne doit attendre l'autre pour traverser le lien. Le coût d'une tâche ne correspond donc pas à sa date d'arrivée mais plutôt à un coût de partage de ressources dans un réseau.

7.1.2 Résumé des résultats obtenus

Nous considérons par la suite les deux problèmes suivants : le problème de la *Date de Fin Maximum*, dans lequel on souhaite minimiser la date à laquelle tous les paquets sont arrivés, et le problème de la *Date de Fin Moyenne*, dans lequel on souhaite minimiser la date d'arrivée moyenne des paquets. Puisque la date d'arrivée moyenne des paquets est égale à la somme des dates d'arrivée (de tous les paquets) divisée par le nombre de paquets, ce dernier problème est équivalent au problème où l'on cherche à minimiser la somme des dates d'arrivée de tous les paquets.

Nous étudions la perte de performance quand nous avons n tâches de différentes longueurs, qui partent d'une même source en même temps, chaque tâche ayant sa propre destination. Les liens du réseau utilisent la même politique qui est SPT, LPT, LRD ou LRT. Pour chacune de ces politiques, nous étudions le rapport d'approximation ou le prix de l'anarchie pour le problème de la Date de Fin Maximum et pour le problème de la Date de Fin Moyenne.

Dans la section 7.2 nous étudions le rapport d'approximation de chaque politique quand le réseau est un chemin ou un arbre. Dans la section 7.3 nous étudions le prix de l'anarchie pour chacune des politiques dans le cas où le réseau est un anneau. Dans la section 7.4, cette perte de performance est étudiée dans le cas où toutes les tâches ont la même source et la même destination, dans un réseau qui est un chemin ou un anneau. Dans la suite de cette section, nous donnons un résumé des résultats obtenus, et nous introduisons quelques notations utilisées dans la suite de ce chapitre.

Le tableau 7.1 (respectivement le tableau 7.2) indique le rapport d'approximation (ou prix de l'anarchie) quand les tâches ont des destinations différentes (respectivement les tâches ont la même destination). Nous donnons soit des valeurs exactes (par exemple pour la politique SPT dans un arbre), soit une borne inférieure et une borne supérieure (par exemple pour la politique SPT dans un anneau).

Chaque politique peut avoir une *sous-politique* pour ordonnancer ses tâches quand il y a un conflit. Par exemple la politique SPT peut ordonnancer les tâches qui ont la même longueur avec la politique LRD. Il est important de remarquer que les résultats donnés dans les tableaux suivants sont valables *quelque soit* la sous-politique de chaque politique. Ainsi, dans le cas où toutes les tâches ont la même destination, nous n'avons pas donné de résultat pour la politique LRD pour le problème de la Date de Fin Moyenne, car le rapport de LRD dépend dans ce cas là de sa sous-politique (il est de 1 si sa sous-politique est SPT, alors qu'il est en $\Theta(n)$ si sa sous-politique est LPT par exemple).

Politique	Chemin		Arbre		Anneau	
	Moyenne	Maximum	Moyenne	Maximum	Moyenne	Maximum
SPT	1	2	1	2	$1.34 < x \leq 2$	$2 \leq x < 3$
LPT, LRT	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Omega(n)$	$\Theta(n)$
LRD	$\Theta(n)$	1	$\Theta(n)$	2	$\Theta(n)$	$1.5 \leq x < 3$

Tab. 7.1 – Résumé des résultats quand les tâches ont des destinations quelconques.

Politique	Chemin et Arbre		Anneau	
	Moyenne	Maximum	Moyenne	Maximum
SPT	1	1	$1.17 < x \leq 2$	$1.66 < x \leq 2$
LPT, LRT	$\Theta(n)$	1	$\Theta(n)$	$1.16 < x \leq 2$
LRD	(voir sous-politique)	1	(voir sous-politique)	$x \leq 2$

TAB. 7.2 – Résumé des résultats quand les tâches ont toutes la même destination.

Notations. Nous adoptons dans la suite de ce chapitre les notations venant du domaine de l'ordonnancement, et que nous avons utilisées dans les chapitres précédents. Ainsi, nous utilisons le terme tâche au lieu de paquet. La longueur d'une tâche i est notée l_i, et le nombre d'arcs (ou liens) du chemin le plus court entre la source et la destination de i est noté x_i. Ainsi, le temps de parcours d'une tâche i qui emprunte le chemin le plus court est égal à $x_i\, l_i$. La date de fin (ou date d'arrivée) d'une tâche i est égale au temps de parcours de i, plus les temps d'attente que la tâche i a éventuellement passé dans des buffers avant d'arriver à sa destination. La date de fin de la tâche i est notée C_i, et la date de fin maximum (i.e. la date de fin de la tâche qui arrive à destination en dernier) est notée C_{max}. A_i représente l'arc que la tâche i emprunte à partir de la source, et $\mathcal{B}_i$ est l'ensemble des tâches qui traversent l'arc A_i avant la tâche i.

7.2 Chemins et arbres : cas où il y a plusieurs destinations

7.2.1 La politique SPT (la plus petite longueur en premier)

Théorème 7.1 *Le rapport d'approximation de la politique SPT dans un arbre, pour le problème de la Date de Fin Maximum, est inférieur à 2.*

Preuve : Soit OPT la date de fin maximum dans une solution optimale de ce problème. Nous considérons une tâche i quelconque et nous montrons que la date de fin de cette tâche, C_i, est inférieure à $2\,OPT$. Puisque nous sommes dans un arbre, il y a un seul chemin possible pour chaque couple (source, destination). De plus, plus une tâche est petite, plus elle quitte la source tôt : ainsi, une fois que la tâche i est partie elle ne rattrapera pas de tâches (les tâches parties avant elle étant de longueurs plus petites, elles sont soit devant i, soit arrivées à destination), et ne sera pas, de la même manière, rattrapée par d'autres tâches. La date de fin de la tâche i est donc égal au temps que i a attendu avant son départ de la source, plus le temps nécessaire à i pour rejoindre sa destination une fois qu'elle a quitté la source. Soit W_i la durée d'attente de i avant son départ de la source. Nous avons donc : $C_i = W_i + (x_i\, l_i)$.
W_i est inférieure à la somme de toutes les tâches (i inclue) qui traversent le même arc que i à la sortie de la source (i.e. qui empruntent l'arc A_i), et OPT est nécessairement supérieure ou égale à cette somme (car dans toute solution ces tâches doivent traverser l'arc A_i, et le temps nécessaire pour cela est égal à la somme des longueurs de ces tâches). Donc $W_i < OPT$. De même, OPT doit être supérieure ou égale au temps nécessaire pour que la tâche i atteigne sa destination (sans

temps d'attente) : $x_i\, l_i \leq OPT$. Ainsi $C_i < 2\,OPT$, et, i étant une tâche quelconque nous en déduisons que $C_{max} < 2\,OPT$. □

Théorème 7.2 *Soit* $\varepsilon > 0$. *Le rapport d'approximation de la politique SPT dans un chemin, pour le problème de la Date de Fin Maximum, est supérieur à* $2 - \varepsilon$.

Preuve : Soit $0 < \varepsilon < 1$. Montrons que le rapport d'approximation de SPT est supérieur à $2 - \varepsilon$ en considérant l'instance suivante : un chemin constitué de $n = \lceil \frac{2}{\varepsilon} \rceil$ arcs. La source est le sommet 0, et chaque sommet $i < n$ a un lien sortant vers le sommet $i + 1$. Il y a $(n-1)$ tâches de longueur $1 - 1/n$, et une tâche t de longueur 1. La destination des tâches de longueur $1 - 1/n$ est le sommet 1, et la destination de la tâche t est le sommet n.
Dans une solution optimale, t est ordonnancée en premier et la date de fin maximum est $OPT = n$. Si la politique de chaque arc est SPT, alors la tâche t sera ordonnancée en dernière position, et sa date de fin sera égale à son temps d'attente avant le départ de la source ($(n-1)(1-1/n)$) plus le temps nécessaire pour atteindre sa destination (n), c'est à dire $(n-1)(1-1/n)+n$. Le rapport d'approximation de la politique SPT est donc dans ce cas égal à $\frac{(n-1)(1-1/n)+n}{n} = 2 - \frac{2}{n} + \frac{1}{n^2} > 2 - \varepsilon$. □

Nous déduisons des théorèmes 7.1 et 7.2 le corollaire suivant :

Corollaire 7.1 *Le rapport d'approximation de la politique SPT dans un chemin ou un arbre, pour le problème de la Date de Fin Maximum, tend vers 2.*

Montrons maintenant que la politique SPT est optimale pour le problème de la Date de Fin Moyenne.

Théorème 7.3 *La politique SPT est optimale pour le problème de la Date de Fin Moyenne dans un arbre.*

Preuve : Nous rappelons que le problème qui consiste à minimiser la date de fin moyenne des tâches est équivalent au problème qui consiste à minimiser la somme des dates de fin des tâches, car la date de fin moyenne des tâches est égale à la somme des dates de fin de toutes les tâches, divisée par le nombre de tâches. Comme le routage pour chaque arc sortant de la source est indépendant, nous allons considérer la somme des dates de fin des tâches empruntant l'arc A_i, et nous allons montrer que cette somme est égale à la somme des dates de fin empruntant l'arc A_i dans une solution optimale. Supposons qu'il y ait n tâches qui empruntent l'arc A_i.
Si la politique de chaque arc est SPT, alors le seul temps d'attente de la tâche i est son temps d'attente avant de partir de la source, et sa date de fin est égale à son temps d'attente avant le départ, que l'on note $W_i(SPT)$, plus le temps dont i a besoin pour atteindre sa destination ($x_i\, l_i$). La somme des dates de fin quand la politique des arcs est SPT est donc égale à $\sum_{i=1}^{n} W_i(SPT) + \sum_{i=1}^{n} x_i\, l_i$.
Considérons maintenant une solution optimale $\mathcal{O}$ pour le problème de la Date de Fin Moyenne. Dans cette solution, la somme des dates de fin est supérieure ou égale à $\sum_{i=1}^{n} W_i(\mathcal{O}) + \sum_{i=1}^{n} x_i\, l_i$, où $W_i(\mathcal{O})$ est le temps que i doit attendre avant son départ dans $\mathcal{O}$, i.e. $W_i(\mathcal{O})$ est égal à la somme des longueurs des tâches ordonnancées sur A_i avant i. Supposons que, dans $\mathcal{O}$, les tâches soient ordonnancées sur A_i dans l'ordre croissant de leurs numéros d'identifications : la tâche 1

est ordonnancée avant la tâche 2, etc, et la tâche n est ordonnancée en dernier. Nous avons alors $\sum_{i=1}^{n} W_i(\mathcal{O}) = 0 + l_1 + (l_1 + l_2) + \cdots + (l_1 + \cdots + l_{n-1}) = (n-1)\, l_1 + (n-2)\, l_2 + \cdots + l_{n-1}$. Cette somme est minimisée si $l_1 \leq l_2 \leq \cdots \leq l_n$, et dans ce cas là $W_i(\mathcal{O}) = W_i(SPT)$. Donc la somme des dates de fin des tâches empruntant l'arc A_i est optimale si la politique des liens est SPT. Puisque ceci est vrai quelque soit l'arc sortant de la source, la somme totale des dates de fin est minimisée quand la politique de chaque arc est SPT. □

7.2.2 La politique LPT (la plus grande longueur en premier)

Théorème 7.4 *Soit n le nombre de tâches à router. Le rapport d'approximation de la politique LPT dans un chemin ou un arbre, pour le problème de la Date de Fin Maximum, est en $\Theta(n)$.*

Preuve : Nous montrons tout d'abord que le rapport d'approximation de la politique LPT dans un chemin (et donc dans un arbre) est en $\Omega(n)$. Pour cela, nous considérons l'instance suivante : nous avons n tâches, et un chemin constitué de $m = 2^{n-1} + 1$ arcs. La source est le sommet 0 et chaque sommet $i < m$ a un lien sortant vers le sommet $i + 1$. Nous avons n tâches $\{1, \ldots, n\}$, et chaque tâche i a une longueur égale à $1/2^{i-1}$, et sa destination est le sommet $(2^{i-1} + 1)$.
Soit OPT la date de fin maximum dans une solution optimale, et $C_{max}(SPT)$ (respectivement $C_{max}(LPT)$) la date de fin maximum si la politique des arcs est SPT (respectivement si la politique des arcs est LPT). Par définition, $OPT \leq C_{max}(SPT)$. Montrons maintenant que $C_{max}(SPT) \leq 3$. Nous montrerons ensuite que la date de fin maximum quand la politique des liens est LPT est en $\Omega(n)$.
Si la politique des liens est SPT, alors le seul temps d'attente pour chaque tâche est son temps d'attente à la source. Le temps d'attente maximum est celui de la plus grande tâche (la tâche 1), et est égal à $\sum_{i=2}^{n} (1/2^{i-1}) < 1$. Nous savons que $C_{max}(SPT)$ est inférieur ou égal au temps d'attente maximum (inférieur à 1), plus le temps maximal dont une tâche a besoin pour atteindre sa destination. Le temps de parcours maximum est 2 (c'est celui de la plus grande tâche), et donc $C_{max}(SPT) < 3$.
Dans le cas où la politique des arcs est LPT, une tâche ne peut pas dépasser une tâche de longueur supérieure à elle. Ainsi, quand la tâche i est arrivée à sa destination, la tâche $i+1$ a encore $2^{i-1}+1$ arcs à traverser (elle avait $2^i + 1$ arcs à traverser et a déjà traversé 2^{i-1} arcs). Ceci représente un temps de parcours égal à $(2^{i-1} + 1)\, l_{i+1} = (2^{i-1} + 1) \times (1/2^i) > 1/2$. Puisque la tâche 1 atteint sa destination au temps 2, nous en déduisons que $C_{max}(LPT) > 2 + \frac{1}{2}\,(n-1)$. Le rapport d'approximation de la politique LPT est donc $\frac{C_{max}(LPT)}{OPT} > \frac{2}{3} + \frac{n-1}{6}$.
Nous avons montré que le rapport d'approximation de LPT est en $\Omega(n)$. Montrons maintenant qu'il est en $O(n)$. La solution obtenue quand la politique des arcs est LPT n'est pas pire que la solution que l'on obtient quand la politique des arcs est LPT et si chaque tâche ne part de la source que quand les tâches précédentes sont arrivées à destination (i.e. les tâches partent de la source par ordre de longueurs décroissantes, et une tâche ne part de la source qu'au moment où la tâche précédente est arrivée à sa destination). Puisque le temps de parcours de chaque tâche (sans temps d'attente) est inférieur ou égal à OPT, la date de fin de la dernière tâche dans cette solution est d'au plus $n\,OPT$. La date de fin de chaque tâche quand la politique des arcs est LPT est donc également d'au plus $n\,OPT$, et le rapport d'approximation de cette politique est donc en $O(n)$.

Étant en $\Omega(n)$ et en $O(n)$, ce rapport est donc en $\Theta(n)$. □

Théorème 7.5 *Soit n le nombre de tâches à router. Le rapport d'approximation de la politique LPT dans un chemin ou un arbre, pour le problème de la Date de Fin Moyenne, est en $\Theta(n)$.*

Preuve : Considérons l'instance suivante : nous avons un arbre qui est un simple chemin constitué de deux arcs, $n-1$ tâches $\{1, \dots, n-1\}$ de longueur 1, et une tâche n de longueur n^3. La source de ces tâches est le sommet 0 ; celui-ci a un lien sortant vers le sommet 1, qui est la destination des petites tâches ; et le sommet 1 a un lien sortant vers le sommet 2, qui est la destination de la tâche n. Dans une solution optimale, la tâche n est ordonnancée après les petites tâches, et la somme des dates de fin est donc $OPT = (\sum_{i=1}^{n-1} C_i) + C_n = n(n-1)/2 + (n - 1 + 2\,n^3) = 2\,n^3 + n^2/2 + n/2 - 1$. Considérons maintenant le cas où la politique des arcs est LPT : la tâche n est ordonnancée en premier et la somme des dates de fin est alors égale à $C_n + \sum_{i=1}^{n-1} C_i = 2\,n^3 + (n-1)\,n^3 + n(n-1)/2 = (n+1)\,n^3 + n^2/2 - n/2$. Le rapport d'approximation de LPT est donc de $\frac{(n+1)n^3+n^2/2-n/2}{2\,n^3+n^2/2+n/2-1}$, ce qui tend vers $(n+1)/2$ quand n tend vers l'infini.

Nous avons montré que le rapport d'approximation de LPT est en $\Omega(n)$. Montrons maintenant qu'il est en $O(n)$. En effet, la date de fin de chaque tâche avec cette politique n'est pas pire que la date de fin qu'elle aurait dans la solution $\mathcal{S}$, où les tâches partent de la source par ordre de longueurs décroissantes, et où une tâche ne part que quand la tâche précédente est arrivée à destination. Nous considérons l'arc sortant de la source qui est emprunté par le plus grand nombre de tâches, et nous supposons qu'il y a k tâches $\{1, \dots, k\}$ de longueurs $l_1 \leq \cdots \leq l_k$, qui empruntent cet arc. Puisque le temps de parcours (sans attente) de chaque tâche i est égal à $l_i\,x_i$, la somme des dates de fin dans $\mathcal{S}$ est égale à $\sum_{i=1}^{k} i\,l_i\,x_i$, alors que la somme des dates de fin dans une solution optimale est supérieure ou égale à $\sum_{i=1}^{k} l_i\,x_i$ (en effet, dans toute solution, la date de fin de la tâche i est d'au moins $l_i\,x_i$). Ainsi, chaque tâche est comptée au plus $k \leq n$ fois plus dans la somme des dates de fin dans $\mathcal{S}$ que dans la somme des dates de fin dans une solution optimale. Ceci est vrai pour toutes les tâches, quelque soit l'arc sortant de la source qu'elles empruntent. Donc la solution $\mathcal{S}$ a un rapport d'approximation en $O(n)$. Puisque la date de fin de chaque tâche quand la politique des machines est LPT est inférieure ou égale à la date de fin de cette tâche dans $\mathcal{S}$, le rapport d'approximation de LPT est également en $O(n)$. Étant en $\Omega(n)$ et en $O(n)$, ce rapport est en $\Theta(n)$. □

7.2.3 La politique LRD (la destination la plus éloignée en premier)

7.2.3.1 Chemin

La politique LRD est optimale pour le problème de la Date de Fin Maximum dans un chemin. En effet, dans [47], A. Hall, S. Hippler, et M. Skutella étudient un problème de flots qui se propagent au cours du temps, dans des graphes orientés dans lesquels le degré sortant de chaque sommet est d'au plus 1, ce qui inclut le cas des chemins. Dans ce problème, chaque flux a une source et un puits (qui sont des sommets du graphe) et une taille (une quantité de données à router). Le but est d'avoir le flot le plus rapide, i.e. de minimiser la date à laquelle tous les flux sont arrivés à leur destination. Les auteurs donnent l'algorithme suivant : quand il y a un conflit entre plusieurs flux utilisant le même arc, l'algorithme donne la priorité au flux qui est le plus loin de sa destination. Ils prouvent que cet algorithme est optimal. Quand la source de chaque flux est la même, cet algorithme correspond à un algorithme distribué dans lequel chaque arc ordonnance le flot (la tâche) dont la distance restante au puits (à sa destination) est la plus grande, i.e. cela correspond au cas où la politique de chaque arc est LRD. La politique LRD est donc optimale pour le problème de la Date de Fin Maximum dans un chemin.

7.2.3.2 Arbre

Théorème 7.6 *Le rapport d'approximation de la politique LRD dans un arbre, pour le problème de la Date de Fin Maximum, est inférieur à 2.*

Preuve : Soit i la tâche qui a la date de fin maximum quand la politique des arcs est LRD, et soit OPT la date de fin maximum dans une solution optimale. Montrons que la date de fin de i, C_i, est inférieure à $2\,OPT$. Si i n'a pas de temps d'attente après son départ de la source, alors C_i est égale au temps de parcours de i, $x_i\,l_i \leq OPT$, plus son temps d'attente avant son départ de la source, $\sum_{j\in\mathcal{B}_i} l_j < OPT$ (la somme des longueurs des tâches qui empruntent un même arc est nécessairement inférieure ou égale à OPT), et donc $C_i < 2\,OPT$.
Considérons maintenant le cas dans lequel la tâche i a au moins un temps d'attente après son départ. Puisque i a un temps d'attente (dans un buffer) après avoir quitté la source, alors cela signifie que i a rattrapé une tâche de $\mathcal{B}_i$, et qu'il y a dans $\mathcal{B}_i$ au moins une tâche qui est plus grande que i (sinon i n'aurait pas rattrapé de tâche). Soit g la plus grande tâche de $\mathcal{B}_i$. Le temps nécessaire à i pour atteindre sa destination est inférieur ou égal à $l_g\,x_i + \sum_{j\in\mathcal{B}_i\setminus g} l_j + l_i$. En effet, la date de fin de i est maximisée quand chaque tâche de $\mathcal{B}_i$ a la même destination que i. Ceci est équivalent à un routage dans un chemin, et donc LRD est optimale dans ce cas là, et la date de fin de i est alors de $l_g\,x_i + \sum_{j\in\mathcal{B}_i\setminus g} l_j + l_i$. En effet, SPT est aussi une politique optimale dans ce cas là (car, étant donné qu'il y a une destination unique, SPT est aussi une politique LRD - elle ordonnance en premier la tâche qui va le plus loin -) et la date de fin maximum de ces tâches avec la politique SPT est égale au temps d'attente de la tâche g, $\sum_{j\in\mathcal{B}_i\setminus g} l_j + l_i$, plus son temps de parcours, $l_g\,x_i$: la date de fin maximum est donc $l_g\,x_i + \sum_{j\in\mathcal{B}_i\setminus g} l_j + l_i$. La somme des longueurs des tâches qui traversent un même arc est inférieure ou égale à OPT, et donc $\sum_{j\in\mathcal{B}_i\setminus g} l_j + l_i < OPT$. De plus, puisque g a quitté la source avant i avec la politique LRD, nous savons que $x_g \geq x_i$ et donc $l_g\,x_i \leq l_g\,x_g \leq OPT$. D'où $C_i < 2\,OPT$, et le rapport d'approximation de la politique LRD est inférieur à 2. □

Théorème 7.7 *Soit $\varepsilon > 0$. Le rapport d'approximation de la politique LRD dans un arbre, pour le problème de la Date de Fin Maximum, est supérieur à $2 - \varepsilon$.*

Preuve : Soit $k = \lceil \frac{2}{\varepsilon} \rceil$. Ceci est un nombre constant, car ε est fixé. Soit n un nombre entier positif tel que n soit un multiple de k, et $n >> k$. Considérons l'instance suivante : l'arbre dessiné sur la figure 7.3, $n - (n/k)$ tâches $\{1, \ldots, (n - n/k)\}$ de longueurs 1 et une tâche t de longueur n/k. Sur la figure 7.3 la source est le sommet S, et d_i indique la destination de la tâche i. Pour chaque $i \in \{1, \ldots, (n - n/k)\}$, la distance entre S et d_i est $k + 1$, alors que la distance entre S et d_t est k. Dans une solution optimale, la tâche t est ordonnancée en premier et la date de fin

maximum, OPT, est égale à $n + k$ (c'est la date de fin de la dernière tâche de longueur 1). Si les politiques des liens sont LRD, alors la tâche t est ordonnancée après les autres tâches et la date de fin maximum est $C_{max}(LRD) = 2\,n - (n/k)$. Le rapport d'approximation de LRD est alors $\frac{C_{max}(LRD)}{OPT} = \frac{2\,n-(n/k)}{n+k} = \frac{(2-1/k)\,(n+k)-(2-1/k)\,k}{n+k} = (2 - \frac{1}{k}) - \frac{2\,k-1}{n+k}$.
Puisque $k = \lceil \frac{2}{\varepsilon} \rceil$, le rapport d'approximation de LRD est égal à $(2 - \frac{1}{\lceil 2/\varepsilon \rceil}) - \frac{2\,\lceil 2/\varepsilon \rceil - 1}{n+\lceil 2/\varepsilon \rceil}$, ce qui tend vers $2 - \frac{1}{\lceil 2/\varepsilon \rceil} \geq 2 - \frac{\varepsilon}{2} > 2 - \varepsilon$ quand n tend vers l'infini. □

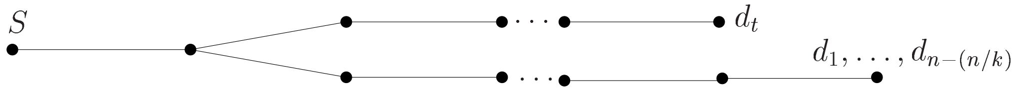


FIG. 7.3 – Exemple dans lequel le rapport d'approximation de la politique LRD tend vers 2 pour le problème de la Date de Fin Maximum.

Nous déduisons des théorèmes 7.6 et 7.7 le corollaire suivant :

Corollaire 7.2 *Le rapport d'approximation de la politique LRD dans un arbre, pour le problème de la Date de Fin Maximum, tend vers* 2.

Nous montrons maintenant que le rapport d'approximation de la politique LRD pour le problème de la Date de Fin Moyenne dans un chemin ou un arbre, n'est pas borné par une constante.

Théorème 7.8 *Soit n le nombre de tâches à router. Le rapport d'approximation de la politique LRD dans un chemin ou un arbre, pour le problème de la Date de Fin Moyenne, est en* $\Theta(n)$.

La preuve est la même que celle du théorème 7.5.

7.2.4 La politique LRT (le plus long temps de parcours en premier)

Théorème 7.9 *Soit n le nombre de tâches à router. Le rapport d'approximation de la politique LRT dans un chemin ou un arbre, pour le problème de la Date de Fin Maximum, est en* $\Theta(n)$.

La preuve est la même que celle du théorème 7.4.

Théorème 7.10 *Soit n le nombre de tâches à router. Le rapport d'approximation de la politique LRT dans un chemin ou un arbre, pour le problème de la Date de Fin Moyenne, est en* $\Theta(n)$.

La preuve est la même que celle du théorème 7.5.

7.3 Anneau : cas où il y a plusieurs destinations

Nous considérons ici un anneau ayant $m \geq 2$ sommets et dans lequel il y a entre deux sommets voisins u et v un arc (u, v) et un arc (v, u). Ainsi, contrairement à ce qui se passe dans le cas des arbres, plusieurs chemins sont possibles entre une source et une destination, et chaque tâche doit

choisir à la source un des deux arcs sortants. Chaque tâche va choisir l'arc qui va lui permettre de minimiser sa date de fin. Avec les politiques que nous étudions (SPT, LPT, LRT et LRD), aucune tâche n'a intérêt à traverser un arc (u, v) puis à retourner à u, le sommet dont elle vient. Une fois qu'une tâche a choisi quel chemin emprunter pour rejoindre sa destination, i.e. en passant par le côté gauche (dans le sens des aiguilles d'une montre), ou par le côté droit de l'anneau (dans le sens inverse des aiguilles d'une montre), alors cette tâche ne changera pas de direction.
Avec la politique LRD, chaque arc ordonnance les tâches par ordre décroissant de leur distance restante à leur destination. La distance restante d'une tâche traversant un arc de la gauche vers la droite (respectivement de la droite vers la gauche) est alors, pour cet arc, le nombre d'arcs que la tâche devra traverser pour arriver à sa destination en empruntant le côté droit (respectivement gauche) de l'anneau (i.e. c'est le nombre d'arcs que la tâche devra traverser si elle ne change pas de direction).

7.3.1 Corollaires des résultats dans les arbres

Le prix de l'anarchie avec une politique donnée dans un anneau est supérieur ou égal au rapport d'approximation de cette politique dans un chemin. En effet, dans le cas où nous avons un très grand anneau dans lequel la destination de chaque tâche est, par exemple, sur le côté droit de l'anneau, alors, dans un équilibre de Nash aussi bien que dans une solution optimale, toutes les tâches vont emprunter le côté droit de l'anneau pour rejoindre leur destination, et l'anneau peut alors être vu comme un chemin. Nous pouvons donc conclure que le prix de l'anarchie avec la politique SPT est supérieur ou égal à 2 pour le problème de la Date de Fin Maximum (cf. théorème 7.1). De même, le prix de l'anarchie des politiques LPT, LRD et LRT est en $\Omega(n)$ pour le problème de la Date de Fin Moyenne (cf. théorème 7.5), et le prix de l'anarchie des politiques LPT et LRT est en $\Omega(n)$ pour le problème de la Date de Fin Maximum (cf. théorème 7.4).

Nous montrons dans les sections suivantes les résultats qu'il reste à prouver pour compléter le tableau 7.1 donné page 124.

7.3.2 Les politiques LPT et LRT

Nous avons montré dans la section précédente que le prix de l'anarchie avec les politiques LPT et LRT est en $\Omega(n)$, où n est le nombre de tâches dans le réseau. Nous montrons maintenant que le prix de l'anarchie est en fait en $\Theta(n)$.

Théorème 7.11 *Soit n le nombre de tâches à router. Le prix de l'anarchie avec la politique LPT, pour le problème de la Date de Fin Maximum, est en $\Theta(n)$.*

Preuve : Nous avons montré dans la section précédente que le prix de l'anarchie avec la politique LPT est en $\Omega(n)$, nous montrons maintenant que ce prix de l'anarchie est en $O(n)$. Supposons que nous ayons n tâches $\{1, \ldots, n\}$ telles que $l_1 \geq l_2 \geq \cdots \geq l_n$. Soit OPT la date de fin maximum dans une solution optimale. Soit C_i la date de fin de la tâche i avec la politique LPT. Nous prouvons ce théorème par récurrence. Pour cela, nous montrons que la date de fin de chacune des j premières tâches (les tâches 1 à j) est inférieure ou égale à $j\,OPT$, et ce pour tout entier j tel que $1 \leq j \leq n$.

Ceci est vrai quand j est égal à 1 : la tâche 1, qui est la plus grande, est ordonnancée en premier quel que soit le chemin qu'elle prenne et elle choisit donc le chemin le plus court pour aller à sa destination. Sa date de fin est $x_1\, l_1 \leq OPT$.

Soit j un entier tel que $1 \leq j < n$. Supposons que la date de fin de chacune des j premières tâches est inférieure ou égale à $j\, OPT$, et montrons que la date de fin de la tâche $j+1$ est inférieure ou égale à $(j+1)\, OPT$.
Puisque toutes les tâches qui précèdent la tâche $j+1$ sont arrivées à la date $j\, OPT$, à partir de cette date, la tâche $j+1$ n'a plus de temps d'attente. Le temps de parcours de la tâche $j+1$ est de $x_{j+1}\, l_{j+1} \leq OPT$ si elle prend son plus court chemin. Si elle prend son plus court chemin la tâche $j+1$ arrivera donc au plus tard en $(j+1)\, OPT$. Si la tâche $j+1$ choisit d'emprunter l'autre chemin, alors elle finira également avant la date $(j+1)\, OPT$, sinon elle aurait emprunté son plus court chemin.

La date de fin maximum est donc inférieure ou égale à $n\, OPT$, et le prix de l'anarchie avec la politique LPT est en $O(n)$. □

Théorème 7.12 *Soit n le nombre de tâches à router. Le prix de l'anarchie avec la politique LRT, pour le problème de la Date de Fin Maximum, est en $\Theta(n)$.*

Preuve : Nous avons montré dans la section précédente que le prix de l'anarchie avec la politique LRT est en $\Omega(n)$, nous montrons maintenant que ce prix de l'anarchie est en $O(n)$. Supposons que nous ayons n tâches $\{1, \ldots, n\}$ telles que $l_1 \geq l_2 \geq \cdots \geq l_n$. Soit OPT la date de fin maximum dans une solution optimale. Soit C_i la date de fin de la tâche i avec la politique LRT. Nous prouvons ce théorème par récurrence. Pour cela, nous montrons que la date de fin de chacune des j premières tâches (les tâches 1 à j) est inférieure ou égale à $(3\, j)\, OPT$, et ce pour tout entier j tel que $1 \leq j \leq n$.

Ceci est vrai quand j est égal à 1 : la tâche 1, qui est la plus grande, a, à la source, un temps d'attente inférieur à $\sum_{i=2}^{n} l_i < 2\, OPT$, et a ensuite un temps de parcours, si elle prend son plus court chemin, égal à $x_1\, l_1 \leq OPT$. Comme la tâche 1 est la plus grande des tâches, les tâches qui partent avant elles de la source sont plus petites, et la tâche 1 ne les rattrape pas ensuite. Sa date d'arrivée est donc inférieure à $3\, OPT$ (si elle n'emprunte pas son plus court chemin alors sa date d'arrivée est aussi inférieure ou égale à $3\, OPT$ sinon elle aurait intérêt à emprunter son plus court chemin).

Soit j un entier tel que $1 \leq j < n$. Supposons que la date de fin de chacune des j premières tâches soit inférieure ou égale à $(3\, j)\, OPT$, et montrons que la date de fin de la tâche $j+1$ est inférieure ou égale à $3\, (j+1)\, OPT$.
Puisque toutes les tâches plus grandes que la tâche $j+1$ sont arrivées à la date $(3\, j)\, OPT$, à partir de cette date, le seul temps d'attente que la tâche $j+1$ peut avoir à subir est égal à $\sum_{i=j+2}^{n} l_j < 2\, OPT$. Si elle prend son plus court chemin, le temps de parcours de la tâche $j+1$ est égal à $x_{j+1}\, l_{j+1} \leq OPT$ et la tâche $j+1$ arrivera donc au plus tard en $(3\, j)\, OPT + 2\, OPT + OPT = 3\, (j+1)\, OPT$. Si elle prend l'autre chemin alors elle arrivera également au plus tard en $3\, (j+1)\, OPT$, car sinon elle emprunterait son plus court chemin.

La date de fin maximum est donc inférieure ou égale à $(3\, n)\, OPT$, et le prix de l'anarchie avec la politique LRT est en $O(n)$. □

7.3.3 La politique SPT

Théorème 7.13 *Le prix de l'anarchie avec la politique SPT, pour le problème de la Date de Fin Maximum, est inférieur à 3.*

Preuve : Une fois qu'une tâche est partie de la source, elle ne peut pas rattraper une autre tâche qui a pris la même direction avant elle, puisque la politique des arcs est SPT (la plus petite tâche, qui est la plus rapide pour traverser un arc, est ordonnancée en premier). Donc une tâche n'a pas de temps d'attente après son départ. Soit i une tâche dont la date de fin est égale à la date de fin maximale C_{max}. Montrons que C_{max} est inférieure à $3\,OPT$, où OPT est la date de fin maximum dans une solution optimale. Le temps d'attente maximal de i est égal à la somme des longueurs de toutes les autres tâches, qui est inférieure à la somme des longueurs de toutes les tâches (i y compris), $\sum_{j=1}^{n} l_j$, qui est inférieure ou égale à $2\,OPT$ car dans une solution optimale la somme des longueurs des tâches ordonnancées sur l'un des deux arcs sortants de la source est d'au moins $(\sum_{j=1}^{n} l_j)/2$ (et donc $OPT \geq (\sum_{j=1}^{n} l_j)/2$). Chaque tâche, connaissant les stratégies des autres tâches, choisit d'aller à sa destination par la gauche de l'anneau ou par la droite de l'anneau, et décide de prendre la direction qui minimise sa date de fin. Si elle choisit le chemin le plus court, de longueur x_i, alors le temps de parcours de la tâche i sera égal à $x_i\, l_i \leq OPT$, et donc sa date de fin sera inférieure à $3\,OPT$. Puisque i souhaite minimiser sa date de fin, elle va choisir d'emprunter l'autre côté de l'anneau seulement si cela n'augmente pas sa date de fin, et donc $C_{max} < 3\,OPT$. $\square$

Théorème 7.14 *Le prix de l'anarchie avec la politique SPT, pour le problème de la Date de Fin Moyenne, est inférieur ou égal à 2.*

Preuve : Supposons que nous devons router n tâches $T = \{1, 2, \ldots, n\}$ de longueurs $l_1 \leq l_2 \leq \cdots \leq l_n$. Soit $\mathcal{O}$ une solution optimale de cette instance pour le problème de la Date de Fin Moyenne. Soit (A, B) une partition des tâches de T dans laquelle A est l'ensemble des tâches qui empruntent l'arc gauche de la source dans $\mathcal{O}$, et B est l'ensemble des tâches qui empruntent l'arc droit de la source dans $\mathcal{O}$. Soit $\mathcal{S}$ un équilibre de Nash quand la politique des arcs est SPT. Soit $\mathcal{S}'$ la solution dans laquelle toutes les tâches prennent le même arc sortant de la source (par exemple l'arc gauche de la source) et quand la politique des arcs est SPT. Soit $W_i(\mathcal{S}')$ (respectivement $W_i(\mathcal{O})$) la date à laquelle la tâche i a traversé le premier arc (l'arc sortant de la source) dans $\mathcal{S}'$ (respectivement dans $\mathcal{O}$). Dans $\mathcal{S}$, la date de fin de chaque tâche i est $C_i(\mathcal{S}) \leq W_i(\mathcal{S}') + (x_i - 1)l_i$. En effet, dans $\mathcal{S}$, chaque tâche i choisit le chemin qui minimise sa date de fin, et en choisissant d'emprunter le chemin le plus court, i aurait une date de fin égale au temps nécessaire pour traverser le premier arc (i.e. le temps d'attente avant son départ plus sa longueur l_i), qui est inférieur ou égal à $W_i(\mathcal{S}')$, plus son temps de parcours une fois qu'elle a traversé le premier arc, qui est égal à $(x_i - 1)l_i$ (nous rappelons qu'une fois qu'une tâche est partie de la source elle n'a plus de temps d'attente avec la politique SPT).

Dans $\mathcal{O}$, la date de fin de i est égale au temps dont elle a besoin pour traverser le premier arc, $W_i(\mathcal{O})$, plus son temps de parcours restant pour aller à sa destination, qui est supérieur ou égal à $(x_i - 1)\, l_i$. Puisque $C_i(\mathcal{S}) \leq W_i(\mathcal{S}') + (x_i - 1)l_i$ et la date de fin de i dans $\mathcal{O}$ est $C_i(\mathcal{O}) \geq W_i(\mathcal{O}) + (x_i - 1)l_i$, si $\sum_{i=1}^{n} W_i(\mathcal{S}') \leq 2 \sum_{i=1}^{n} W_i(\mathcal{O})$, alors nous pouvons en déduire que la somme des dates de fin dans $\mathcal{S}$ est inférieure ou égale à la somme des dates de fin dans $\mathcal{O}$. Montrons maintenant que $\sum_{i=1}^{n} W_i(\mathcal{S}') \leq 2 \sum_{i=1}^{n} W_i(\mathcal{O})$.

Tout d'abord, la somme des dates auxquelles les tâches ont traversé le premier arc dans $\mathcal{S}'$ est égale à $n\,l_1 + (n-1)\,l_2 + \cdots + 2\,l_{n-1} + l_n$. En effet, le temps d'attente de la tâche i dans $\mathcal{S}'$ est égal à $l_1 + \cdots + l_{i-1}$, et donc la date à laquelle la tâche i a traversé le premier arc est égale à $\sum_{j=1}^{i} l_j$. Puisque la tâche i est, dans la file d'attente pour être routée sur le premier arc, devant $n-i$ autres tâches, l_i sera comptée $n-i+1$ fois dans la somme des dates auxquelles les tâches ont traversé le premier arc (elle sera comptée une fois dans le temps d'attente de chacune des $n-i$ tâches derrière elle, et une fois pour le temps dont i a besoin pour traverser le premier arc).
Nous avons vu qu'il y a dans $\mathcal{O}$ deux files A et B. Regardons la plus petite valeur que $s = \sum_{i=1}^{n} W_i(\mathcal{O})$ peut prendre. Dans chaque file F la tâche au $i^{\text{ème}}$ rang (i.e. routée après $i-1$ tâches) sera comptée $|F|-i$ fois dans la somme des temps d'attente des autres tâches, et donc $|F|-i+1$ dans s. La dernière tâche de chaque file sera comptée seulement une fois, l'avant dernière tâche sera comptée deux fois, et ainsi de suite. Donc la seule manière de minimiser s est de compter l_n et l_{n-1} seulement une fois, l_{n-2} et l_{n-3} deux fois, etc. De cette façons $l_{n-(2\,i)}$ et $l_{n-(2\,i+1)}$ seront comptées $i+1$ fois dans s, alors qu'elles sont comptées $(2i+1)$ et $(2i+2)$ fois dans $\sum_{i=1}^{n} W_i(\mathcal{S}')$. Donc la tâche i est au plus comptée deux fois plus dans $\sum_{i=1}^{n} W_i(\mathcal{S}')$ que dans la valeur minimum de $\sum_{i=1}^{n} W_i(\mathcal{O})$. Donc $\sum_{i=1}^{n} W_i(\mathcal{S}') \leq 2\,\sum_{i=1}^{n} W_i(\mathcal{O})$ et donc la somme des dates de fin dans $\mathcal{S}$ est inférieure ou égale à deux fois la somme des dates de fin dans $\mathcal{O}$. □

Théorème 7.15 *Le prix de l'anarchie avec la politique SPT, pour le problème de la Date de Fin Moyenne, est supérieur ou égal à* $55/41 \approx 1.34$.

Preuve : Considérons l'instance montrée sur la figure 7.4 : un anneau de longueur 10, et 9 tâches : 6 tâches $\{1, \ldots, 6\}$ de longueur 1 ; 2 tâches $\{7, 8\}$ de longueur $1-\varepsilon$; et une tâche 9 de longueur $1-2\,\varepsilon$, où ε est une petite valeur positive. Dans la figure 7.4, S indique la source et d_i indique la destination de la tâche i.

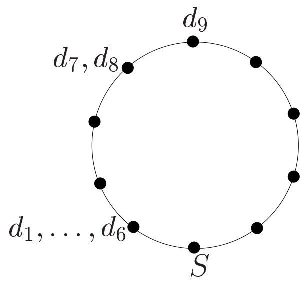


FIG. 7.4 – Exemple dans lequel le prix de l'anarchie avec la politique SPT tend vers $55/41$ pour le problème de la Date de Fin Moyenne.

Si toutes les tâches empruntent le côté gauche de l'anneau, nous avons un équilibre de Nash, et la somme des dates de fin est égale à $55 - 41\,\varepsilon$. Dans une solution optimale, les tâches $7, 8$ et 9 empruntent le côté droit de l'anneau et la somme des dates de fin est égale à $41 - 16\,\varepsilon$. Donc le prix de l'anarchie avec la politique SPT est supérieur ou égal à $\frac{55-41\,\varepsilon}{41-16\,\varepsilon}$, ce qui tend vers $\frac{55}{41}$ quand ε tend vers 0. □

7.3.4 La politique LRD

Théorème 7.16 *Le prix de l'anarchie avec la politique LRD, pour le problème de la Date de Fin Maximum, est inférieur à 3.*

Preuve : Soit T l'ensemble des tâches à router, et (X_1, X_2) une partition de T. Soit (X_1, X_2, Pol) la solution obtenue quand la politique de chaque arc est Pol et quand les tâches qui choisissent d'emprunter l'arc gauche de la source sont dans X_1 et les tâches qui choisissent d'emprunter l'arc droit de la source sont dans X_2. Soit $C_{max}(X_1, X_2, Pol)$ la date de fin maximum dans cette solution, et soit $C_{max}(X, Pol)$ la date de fin maximum dans cette solution parmi les tâches qui appartiennent à l'ensemble X (où X est soit X_1 soit X_2) . Soit $\mathcal{S}$ un équilibre de Nash quand la politique des arcs est LRD, et soit A (respectivement B) l'ensemble des tâches qui a choisi de prendre l'arc gauche (respectivement l'arc droit) de la source dans $\mathcal{S}$.

Supposons sans perte de généralité que la source est le sommet 1, qui a deux voisins : le sommet 2 à sa gauche, et le sommet m à sa droite. Une fois que les tâches ont été partitionnées en deux ensembles A et B, le routage peut être vu comme deux routages parallèles dans deux chemins : un chemin qui commence au sommet 1, qui pointe vers le sommet 2, qui pointe vers le sommet 3, $\ldots$, qui pointe vers le sommet m, et dans lequel les tâches appartenant à A sont routées ; et un chemin qui commence au sommet 1, qui pointe vers le sommet m, $\ldots$, qui pointe vers le sommet 2, et dans lequel les tâches appartenant à B sont routées. LRD est une politique optimale dans un chemin, et donc, étant donnée une affectation des tâches dans les deux buffers gauche et droite, la politique LRD engendre un routage au moins aussi bon que celui obtenu avec les autres politiques : $C_{max}(A, B, LRD) \leq C_{max}(A, B, SPT)$. Comparons maintenant $C_{max}(A, B, SPT)$ à la date de fin maximum OPT d'une solution optimale. Soit i la tâche qui se termine en dernier dans (A, B, SPT) ; nous supposons sans perte de généralité que i appartient à A. Sa date de fin est égale à $C_{max}(A, B, SPT) = W_i + l_i\, D$, où W_i est le temps d'attente de i avant son départ de la source ($W_i < 2\,OPT$), et D est le nombre d'arcs que i doit traverser pour arriver à sa destination. Si le chemin que i prend pour aller à sa destination est le chemin le plus court, alors $l_i\, D \leq OPT$ et $C_{max}(A, B, LRD) \leq C_{max}(A, B, SPT) < 3\,OPT$. Sinon le chemin le plus court de la source à la destination de i est le chemin du côté droit de l'anneau, et i aurait emprunté son plus court chemin si elle avait été dans B. Là, sa date de fin aurait été inférieure à $3\,OPT$ (le temps d'attente est inférieur à $2\,OPT$, et le temps de parcours est inférieur ou égal à OPT). Puisque nous sommes dans un équilibre de Nash, i minimise sa date de fin dans (A, B, LRD) en empruntant le chemin gauche de l'anneau, et sa date de fin est alors inférieure ou égale à la date de fin qu'elle aurait eu en se plaçant dans B, et donc est inférieure ou égale à $C_{max}(B, LRD) \leq C_{max}(B, SPT) < 3\,OPT$.
□

Théorème 7.17 *Le prix de l'anarchie avec la politique LRD, pour le problème de la Date de Fin Maximum, est supérieur ou égal à 1.5.*

Preuve : Considérons l'instance montrée sur la figure 7.5. Nous avons un anneau de longueur $2\,n + 2$, et n tâches : une tâche 1 de longueur $2\,n$, et $n - 1$ tâches $\{2, \ldots, n\}$ de longueur 1. La destination de la tâche i est le $i^{\text{ème}}$ sommet à la gauche de la source. Dans la figure 7.5, S indique la source, et d_i indique la destination de la tâche i. Le nombre à l'intérieur de l'anneau à chaque noeud d_i indique la date de fin de la tâche i.

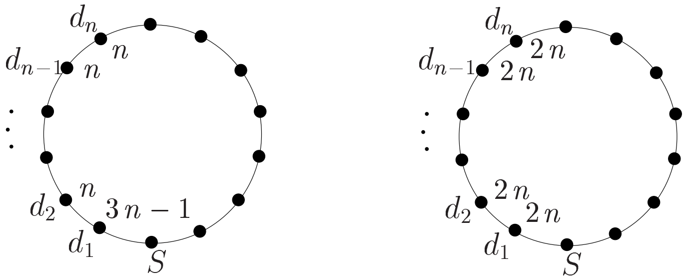


FIG. 7.5 – Exemple dans lequel le prix de l'anarchie tend vers $3/2$ pour le problème de la Date de Fin Maximum avec la politique LRD. *Gauche* : équilibre de Nash. *Droite* : solution optimale.

Dans l'équilibre de Nash montré sur la figure 7.5 *Gauche*, toutes les tâches empruntent le côté gauche de l'anneau et la date de fin maximum est $3\,n-1$. Dans la solution optimale, montrée sur la figure 7.5 *Droite*, la tâche 1 emprunte le côté gauche de l'anneau, les autres tâches empruntent le côté droit de l'anneau, et la date de fin maximum est $2\,n$. Donc le prix de l'anarchie quand la politique des arcs est LRD est ici de $(3\,n-1)/(2\,n)$, ce qui tend vers $3/2$ quand n tend vers l'infini. □

Théorème 7.18 *Soit n le nombre de tâches à router. Le prix de l'anarchie avec la politique LRD, pour le problème de la Date de Fin Moyenne, est en $\Theta(n)$.*

Preuve : Nous avons déjà vu dans la section 7.3.1 que le prix de l'anarchie avec la politique LRD, pour le problème de la Date de Fin Moyenne, est en $\Omega(n)$. Nous montrons donc maintenant que le prix de l'anarchie avec la politique LRD pour ce problème est en $O(n)$. Pour cela nous considérons une instance quelconque du problème de routage dans un anneau. Soit OPT_m la date de fin maximum dans une solution optimale de cette instance pour le problème de la Date de Fin Maximum, et soit OPT_s la somme des dates de fin dans une solution optimale de cette instance pour le problème de la Date de Fin Moyenne. Soit C_{max} la date de fin maximum dans un équilibre de Nash quand la politique des arcs est LRD, et soit S la somme des dates de fin dans ce même équilibre de Nash quand la politique des arcs est LRD. Nous avons montré dans le théorème 7.16 que le prix de l'anarchie avec la politique LRD, pour le problème de la Date de Fin Maximum, est inférieur à 3. Ceci signifie que $C_{max} < 3\,OPT_m$. Par ailleurs, nous savons que $OPT_s > OPT_m$ car OPT_s est la somme des dates de fin, et au moins une tâche a une date de fin supérieure ou égale à OPT_m (par définition de OPT_m). Nous savons de plus que $S \leq n\,C_{max}$, et donc $S \leq n\,(3\,OPT_m) \leq 3\,n\,OPT_s$, ce qui montre que le prix de l'anarchie avec LRD est en $O(n)$. □

7.4 Cas dans lequel la destination des tâches est la même

Nous nous intéressons maintenant au prix de l'anarchie (pour les anneaux), ou au rapport d'approximation (pour les arbres) dans le cas où toutes les tâches ont la même source et la même destination. La politique LRD n'a ici pas d'intérêt si elle n'a pas de sous-politique pour donner une priorité aux tâches qui ont la même destination : chaque politique qui n'introduit pas de temps d'attente (i.e. qui ne laisse pas un arc inactif alors que des tâches attendent d'être routées sur cet arc) est en effet une politique LRD, puisqu'elle ordonnance les tâches de celle qui va le plus loin à celle dont la destination est la plus proche. Nous allons montrer que, dans un anneau, toutes ces politiques ont un prix de l'anarchie de 2 au plus pour le problème de la Date de Fin Maximum. Nous remarquons également que, puisqu'il y a une seule destination, parmi les tâches qui attendent d'être routées sur un même arc, plus une tâche est grande plus son temps de parcours restant est lui aussi grand, et donc les politiques LPT et LRT sont les mêmes dans ce cas là.

7.4.1 Chemin et arbre

Le cas du chemin et de l'arbre est ici le même puisqu'il y a un seul chemin entre une source et une destination dans un arbre. Puisque la politique SPT est optimale pour le problème de la Date de Fin Moyenne quand il y a plusieurs destinations, elle est aussi optimale quand il y a une seule destination. Dans le cas des politiques LPT et LRT et pour ce même problème, nous pouvons utiliser la preuve du théorème 7.5 en la modifiant très légèrement : en considérant que le chemin est composé d'un seul arc sur lequel l'on souhaite router une très grande tâche suivie de nombreuses petites tâches. Le rapport d'approximation est alors en $\Omega(n)$. Le rapport d'approximation de LRD pour ce problème dépend de sa sous-politique : si elle n'a pas de sous-politique, alors LRD peut être la politique la pire possible, par exemple LPT, et son rapport d'approximation est en $\Omega(n)$. Comme par ailleurs le rapport d'approximation des politiques LPT, LRT et LRD est en $O(n)$ quand il y a plusieurs destinations, ceci est bien sûr également vrai quand la destination des tâches est la même.

Pour le problème de la Date de Fin Maximum, nous savons que la politique LRD est optimale dans un chemin (voir section 7.2.3). Puisqu'il y a une seule destination, les politiques SPT, LPT et LRT sont aussi des politiques LRD et sont donc optimales pour ce problème.

7.4.2 Anneau

Le problème dans lequel les tâches ont toutes la même destination est un sous-problème du cas dans lequel les tâches peuvent avoir des destinations différentes, et donc les bornes supérieures du prix de l'anarchie obtenues dans le cas où les destinations sont quelconques sont aussi bien sûr des bornes supérieures du prix de l'anarchie dans le cas où il y a une unique destination. Nous en déduisons ainsi que le prix de l'anarchie pour le problème de la Date de Fin Moyenne est inférieur ou égal à 2 avec la politique SPT. De même, le théorème 7.18 nous indique que le prix de l'anarchie avec la politique LRD est en $O(n)$ (puisqu'en $\Theta(n)$) pour ce problème. Puisque la destination de toutes les tâches est la même, les politiques LPT et LRT sont également des politiques LRD, et nous en déduisons que le prix de l'anarchie est également en $O(n)$ avec les politiques LPT et LRT pour le problème de la date de fin moyenne.

De plus, comme nous l'avons vu dans la section 7.3, le rapport d'approximation d'une politique dans un chemin est une borne inférieure du prix de l'anarchie avec cette politique dans un anneau. Le prix de l'anarchie est donc en $\Omega(n)$ avec les politiques LPT et LRT pour le problème de la Date de Fin Moyenne. Ainsi, étant en $O(n)$ et en $\Omega(n)$, le prix de l'anarchie avec les politiques LPT et LRT est en $\Theta(n)$ pour le problème de la Date de Fin Moyenne.

Si l'anneau est constitué uniquement de deux arcs (i.e. $m = 2$), alors le problème de la Date de Fin Maximum est équivalent au problème dans lequel on veut minimiser la date de fin maximum dans un ordonnancement constitué de deux liens (ou machines) parallèles $(P2||C_{max})$. Nous avons déjà vu, dans les deux chapitres précédents, que le seul équilibre de Nash quand la politique d'un lien est LPT est équivalent à la solution que l'on obtiendrait avec l'algorithme de liste LPT. En effet, supposons que dans un équilibre de Nash une tâche i commence après une tâche j plus petite qu'elle ; afin de diminuer sa date de fin, i a intérêt à aller sur le lien sur lequel j est placée, et cette solution n'est donc pas un équilibre de Nash. Puisque le rapport d'approximation de l'algorithme de liste LPT pour le problème $(P2||C_{max})$, est de $7/6$ [42], le prix de l'anarchie avec la politique LPT est d'au moins $7/6$ pour le problème de la Date de Fin Maximum.

Nous allons maintenant prouver les résultats restants indiqués dans le tableau 7.2 donné page 125. Le théorème suivant montre que le prix de l'anarchie pour le problème de la Date de Fin Maximum est d'au plus 2 pour les politiques SPT, LPT, LRT et LRD.

Théorème 7.19 *Le prix de l'anarchie avec n'importe politique qui n'introduit pas de temps d'inactivité, pour le problème de la Date de Fin Maximum, est inférieur ou égal à 2.*

Preuve : Nous considérons un anneau de taille $m \geq 2$ et n tâches ayant la même source et la même destination. Soit $\mathcal{O}$ une solution optimale de notre problème, et soit OPT la date de fin maximum dans $\mathcal{O}$. Soit (A, B) une partition des tâches dans laquelle A est l'ensemble des tâches qui prennent le chemin gauche de l'anneau (de longueur L_a) dans $\mathcal{O}$, et B est l'ensemble des tâches qui prennent le chemin droit de l'anneau (de longueur L_b) dans $\mathcal{O}$. La longueur du plus court chemin est notée $L = \min\{L_a, L_b\}$. Soit t la plus grande tâche. Nous supposons sans perte de généralité que t appartient à A. Nous avons $OPT \geq \sum_{i \in A \setminus t} l_i + l_t\, L$. En effet, le routage des tâches de A est un problème de routage dans un chemin, problème pour lequel la politique LRD est optimale. Puisque la destination est unique, toutes les politiques qui n'introduisent pas de temps d'inactivité sont optimales, et leur date de fin maximum est la même que celle obtenue avec la politique SPT, c'est à dire la somme des longueurs des tâches avant t, plus le temps de parcours de la tâche t $(l_t\, L_a)$: $\sum_{i \in A \setminus t} l_i + l_t\, L_a \geq \sum_{i \in A \setminus t} l_i + l_t\, L$. De plus, nous savons également que $OPT \geq \sum_{i \in B} l_i$, puisque les tâches de B ont au moins un arc à traverser.

La date de fin maximum dans un équilibre de Nash ne peut pas être pire que la date de fin maximum dans la solution dans laquelle toutes les tâches prennent le plus court chemin : la date de fin maximum est alors égale à $\sum_{i=1, i \neq t}^{n} l_i + l_t\, L$. En effet, si, dans un équilibre de Nash, une tâche a une date de fin supérieure à celle-ci, alors cette tâche est nécessairement routée sur le chemin le plus long, et elle diminuerait sa date de fin en empruntant le chemin le plus court ; ce qui contredit l'hypothèse selon laquelle cette solution est un équilibre de Nash. La date de fin maximale dans un équilibre de Nash est donc inférieure ou égale à $\sum_{i=1, i \neq t}^{n} l_i + l_t\, L = (\sum_{i \in B} l_i) + (\sum_{i \in A \setminus t} l_i + l_t\, L) \leq 2\, OPT$. □

Théorème 7.20 *Le prix de l'anarchie avec la politique SPT, pour le problème de la Date de Fin Maximum, est supérieur ou égal à 5/3.*

Preuve : Considérons l'instance dans laquelle nous avons un anneau de longueur 3, et trois tâches : deux tâches de longueur 1, et une tâche de longueur 3. La destination des tâches est le sommet à la droite de la source : il y a de la source à ce sommet un chemin de longueur 2 et un chemin de longueur 1. La solution dans laquelle toutes les tâches empruntent le chemin de longueur 1 est un équilibre de Nash, et la date de fin maximum est 5. Dans une solution optimale, les tâches de longueur 1 prennent le chemin de longueur 2 et la tâche de longueur 3 prend le chemin de longueur 1 : la date de fin maximum est alors de 3. Le prix de l'anarchie avec la politique SPT pour le problème de la Date de Fin Maximum est donc supérieur ou égal à 5/3. □

Théorème 7.21 *Le prix de l'anarchie avec la politique SPT, pour le problème de la Date de Fin Moyenne, est supérieur ou égal à* $219/187 \approx 1.17$.

Preuve : Considérons l'instance dans laquelle nous avons un anneau de longueur 4, et six tâches : trois tâches de longueur 1, une tâche de longueur 1.5, une tâche de longueur 2.25, et une tâche de longueur 3.375. La destination des tâches est le premier sommet à la droite de la source : il y a de la source à ce sommet un chemin de longueur 3 et un chemin de longueur 1. La solution dans laquelle toutes les tâches empruntent le chemin de longueur 1 est un équilibre de Nash, et la somme des dates de fin est 27.375. Dans une solution optimale, deux tâches de longueur 1 prennent le chemin de longueur 3, et les autres tâches prennent le chemin de longueur 1 : la somme des dates de fin est alors de 23.375. Le prix de l'anarchie avec la politique SPT pour le problème de la Date de Fin Moyenne est donc supérieur ou égal à $27.375/23.375 = 219/187$. □

7.5 Conclusion

Nous avons étudié, dans des réseaux qui sont des arbres ou des anneaux, le rapport d'approximation et le prix de l'anarchie pour des politiques simples qui dépendent de la longueur des tâches et/ou de la distance à leur destination. Certaines bornes inférieures et supérieures ne coïncident pas dans le cas de l'anneau, et une perspective immédiate serait bien sûr d'affiner ces bornes.

Il serait aussi intéressant d'étudier le problème plus général dans lequel la source des paquets n'est pas nécessairement la même. Nous remarquons cependant que dans ce cas le prix de l'anarchie (ou rapport d'approximation) pour le problème de la Date de Fin Moyenne est non borné par une constante (il est en $\Omega(n)$) si les politiques des liens n'introduisent pas de délais. En effet, si on considère le sous-graphe suivant : un sommet 0, qui pointe vers un sommet 1 qui pointe vers un sommet 2, et si nous avons une grande tâche (de taille n^3) qui part de 1 pour aller en 2, et n petites tâches (de taille 1) qui partent de 0 pour aller en 2, alors, quand la première petite tâche arrive au sommet 1 la grande tâche est déjà partie. Toutes les petites tâches doivent donc attendre que la grande tâche soit routée, avant d'emprunter l'arc vers le sommet 2, et le rapport d'approximation est en $\Omega(n)$. Dans ce cas, et si la préemption de tâches est autorisée (i.e. un lien en train de router une tâche peut arrêter de router cette tâche pour en router une autre arrivée entre temps dans son buffer), alors il pourrait être intéressant d'étudier la politique SRPT (pour "Shortest Remaining Processing Time") : chaque lien route la tâche dont la longueur restante est la plus petite (i.e. si une tâche de longueur l_i arrive dans le buffer d'un lien qui est en train de router une tâche dont

il lui reste une longueur supérieure à l_i à router, alors cette tâche sera interrompue - et retournera dans le buffer - au profit de la tâche de longueur l_i qui sera routée à sa place).
De plus, si chaque tâche part de sa propre source, il se pose alors la question de l'information dont dispose une tâche pour effectuer le choix du chemin qu'elle emprunte. Jusqu'à présent, toutes les tâches partant de la même source, il était naturel de considérer qu'une tâche connaissait la stratégie des autres tâches et faisait son choix en fonction de ces stratégies. Si les sources des tâches sont différentes, nous pouvons alors soit considérer qu'une tâche ne connaît les stratégies que des tâches qui partent de la même source qu'elle (nous sommes alors dans le cas d'un jeu à information incomplète), soit qu'une tâche connaît les stratégies de toutes les tâches. Notons que cette deuxième option est la plus souvent utilisée dans la littérature pour des problèmes proches de celui-là (voir par exemple [29, 24, 25]).

Par ailleurs, il serait intéressant d'étudier le prix de l'anarchie dans des graphes généraux. L'étude du prix de l'anarchie dans un graphe (quelconque) dans lequel toutes les tâches ont la même source et la même destination semble notamment un problème intéressant.

Une question importante non abordée dans ce chapitre concerne l'existence d'équilibres de Nash purs. Nous savons que, pour chaque jeu, il existe un équilibre de Nash mixte [63], mais il existe certains jeux pour lesquels il n'existe pas d'équilibre de Nash pur. Par exemple, pour les jeux de congestion, que nous avons évoqués dans la section 7.1.1, il existe toujours un équilibre de Nash pur si toutes les tâches ont la même longueur, comme l'a montré R. Rosenthal dans [71], mais par contre, si les tâches ont des longueurs différentes, alors l'existence d'équilibres de Nash purs est un problème ouvert [24]. De plus, même si un équilibre de Nash pur existe pour toutes les instances, il n'est pas toujours possible de le trouver en un temps polynomial. Or une solution n'est réalisable pratiquement que si les tâches parviennent à un équilibre de Nash en un temps raisonnable. Une question ouverte concerne les politiques LRD et LRT : existe-t'il toujours pour ces politiques un équilibre de Nash pur, et, si oui, est-il possible d'atteindre cet équilibre en un temps raisonnable ? Il est facile de voir qu'il existe, pour les politiques SPT et LPT un équilibre de Nash pur qui peut être atteint très rapidement (en un temps linéaire en le nombre de tâches). Considérons par exemple la politique SPT. La plus petite tâche sait que, quel que soit le chemin qu'elle choisisse, elle sera routée en premier. Elle choisit donc d'emprunter son plus court chemin (ou un de ses plus courts chemins). Connaissant la stratégie de la plus petite tâche, la deuxième plus petite tâche peut alors calculer le chemin le plus court pour aller à sa destination. En effet, cette tâche sera la deuxième à être routée si elle emprunte le même arc que la plus petite tâche, et elle sera la première à être routée sinon. Ainsi, connaissant les stratégies des i plus petites tâches la $(i+1)$-ème petite tâche est capable de déterminer sa stratégie. Le raisonnement est exactement le même avec la politique LPT.

Il est également possible, en modifiant très légèrement les politiques SPT et LPT, de faire en sorte que, avec ces politiques, le seul équilibre de Nash qui existe soit un équilibre de Nash pur. Ceci peut être intéressant si les tâches peuvent adopter une stratégie mixte (i.e. elles ne choisissent pas forcément un seul lien sortant de chaque sommet, mais plusieurs liens, chacun avec une probabilité), car le prix de l'anarchie risque d'être plus important dans ce cas. Il est possible, avec les politiques SPT et LPT, d'avoir des équilibres de Nash mixtes. Par exemple, si la politique des liens est SPT et si la distance à la destination de la plus petite tâche est la même en passant par le côté gauche ou le côté droit de l'anneau, alors la plus petite tâche aura la même date de fin si elle choisit de passer par le côté gauche de l'anneau, ou si elle choisit de passer du côté droit de l'anneau. Il existe donc un équilibre de Nash mixte dans lequel cette tâche choisit d'emprunter avec une probabilité p l'arc gauche de la source et avec une probabilité $1-p$ l'arc droit de la source. Afin que le seul équilibre de Nash possible soit un équilibre de Nash pur, nous pouvons modifier les politiques SPT et LPT de la façon suivante : chaque lien $i \in \{1, \dots, m\}$ de l'anneau ajoute, avant de router la tâche qu'il a choisi de router (cette tâche est choisie suivant la politique SPT ou LPT), un temps d'inactivité égal à $\varepsilon/(2^i)$ (i.e. dès que le lien i a choisi une tâche à router, il attend un temps égal à $\varepsilon/(2^i)$ avant de router cette tâche). Ici, ε est une petite valeur positive telle que $\varepsilon < 1/n$ (où n est le nombre de tâches), et $\varepsilon < \delta/n$ (où δ est la différence minimale entre les longueurs de deux tâches de longueurs différentes). De cette façon, il existe un seul équilibre de Nash et cet équilibre de Nash est pur. Si par exemple la politique est SPT alors la plus petite tâche choisit son chemin (en tenant compte des petits délais), et ce chemin est unique (car il n'y a pas deux sous-ensembles distincts de $\{1/(2^1), \dots, 1/(2^m)\}$ dont les sommes des valeurs soient égales). Les tâches suivantes choisissent leurs chemins de la même façon (en fonction des stratégies adoptées par les tâches plus petites qu'elles). De plus, la somme de tous les petits délais sur un chemin est inférieure à ε, et le prix de l'anarchie augmente donc au plus de $n\,\varepsilon$, ce qui est négligeable car ε peut être fixé aussi petit que l'on veut. Notons que cette technique, utilisée pour obtenir un unique équilibre de Nash pur, marche également si le graphe est un graphe quelconque. De plus, si le graphe est un anneau et que l'on sait par avance quel sommet est la source, alors il suffit qu'un lien sortant de la source ajoute un délai égal à ε avant chaque tâche pour éviter la possibilité d'avoir un équilibre de Nash mixte.

Conclusion

Au cours de cette thèse, nous avons étudié plusieurs problèmes, en informatique théorique, liés au domaine des réseaux. Nous résumons à présent les résultats que nous avons obtenus dans les chapitres précédents.

Résultats obtenus

Dans le **chapitre 2**, nous avons étudié un problème de groupage de trafic dans un réseau optique en étoile. Il est possible, depuis quelques années seulement, de grouper dans un réseau optique plusieurs requêtes dans une même longueur d'onde. Cette possibilité nécessite néanmoins de stocker et traiter ces requêtes au niveau du noeud central, ce qui n'est pas nécessaire lorsque les requêtes sont routées optiquement, i.e. quand elles bénéficient d'une longueur d'onde à elle seules de leur source à leur destination. Notre but était de maximiser la quantité de données routées optiquement. Nous avons montré que ce problème est NP-difficile dans le cas où le nombre de longueurs d'onde est quelconque, et nous avons donné un algorithme polynomial optimal dans le cas où il y a deux longueurs d'onde par fibre. Nous avons donné deux algorithmes d'approximation dans le cas général.

Dans le **chapitre 3**, nous avons étudié un problème NP-difficile de tournées de véhicule. Le but était de minimiser la distance totale qu'un véhicule ayant une capacité donnée doit couvrir afin de servir tous ses clients. La recherche locale donne souvent de bons résultats pour ce genre de problème. Nous nous sommes alors intéressés à un voisinage de taille exponentielle et nous avons montré que ce voisinage peut être exploré en un temps polynomial. Ce travail généralise une étude de G. Gutin [44] pour le même voisinage dans le cas du problème du voyageur de commerce.

Dans le **chapitre 4**, nous nous sommes intéressés au problème qui consiste à ordonnancer des tâches de longueurs différentes sur des machines parallèles. Nous avons étudié des mesures d'équité pour ce problème d'ordonnancement. Tout d'abord, une mesure d'équité entre les machines : nous avons étudié le problème qui consiste à minimiser le nombre moyen de tâches non terminées par machine et par unité de temps. Nous avons montré que ce problème est NP-difficile, mais que l'algorithme de liste SPT a un rapport d'approximation inférieur à 3 pour ce problème. Nous nous sommes également intéressé à deux mesures d'équité entre les tâches (en supposant que chaque tâche souhaite se terminer le plus rapidement possible). Nous avons alors montré que l'algorithme de liste SPT – qui est optimal pour la date de fin moyenne des tâches – possède également de bonnes propriétés d'équité vis-à-vis des tâches puisqu'aucun algorithme ne peut avoir un rapport d'approximation global (ni un rapport de satisfaction individuelle) meilleur que celui de SPT.

Nous avons considéré dans ces premiers chapitres des algorithmes centralisés, et dans lesquels les données ne sont pas détenues par des entités indépendantes de l'algorithme. Nous avons alors considéré dans une deuxième partie des problèmes qui font face à des agents indépendants et individualistes. Notre but a alors été d'optimiser une fonction objectif globale, qui représente le bien-être global de l'ensemble des agents, et cela malgré le comportement des agents (chacun d'entre eux cherchant en effet à optimiser sa propre fonction objectif représentant son propre intérêt).

Dans le **chapitre 5**, nous nous sommes intéressés à un problème d'ordonnancement dans lequel les tâches sont autonomes, puisqu'elles peuvent choisir elles mêmes la machine sur laquelle elles seront ordonnancées. Chaque machine a une politique qui lui indique comment ordonnancer ses tâches, et le but de chaque tâche est de se terminer le plus rapidement possible. Notre but était de minimiser la date de fin de l'ordonnancement. Nous avons pour cela étudié la performance des algorithmes qui proposent un ordonnancement des tâches selon deux critères : la date de fin de l'ordonnancement, et la stabilité de l'ordonnancement (le fait que les tâches aient ait un intérêt le plus faible possible à changer de machine). Nous avons étudié ces performances dans le cadre de trois politiques des liens : LPT, SPT, et une politique probabiliste. Nous avons mis en évidence le compromis nécessaire entre la stabilité et la date de fin de l'ordonnancement avec la politique LPT, et montré que cette politique est toujours meilleure que la politique SPT. Si les tâches ne peuvent changer de machine une fois qu'elles ont choisi une machine, alors nous avons montré qu'il est possible d'avoir un ordonnancement optimal et stable avec la politique qui ordonnance les tâches dans un ordre aléatoire.

Dans le **chapitre 6**, nous nous sommes intéressés à un problème d'ordonnancement dans lequel les tâches sont indépendantes de l'algorithme qui les ordonnance, et les longueurs réelles des tâches ne sont alors pas connues de cet algorithme. Le but de chaque tâche est de se terminer le plus tôt possible, et une tâche peut donc augmenter artificiellement sa longueur si cela diminue sa date de fin. Notre but était alors de concevoir des algorithmes avec véracité garantie, et nous avons considéré deux modèles d'exécution des tâches. Nous avons donné des bornes inférieures (des résultats d'impossibilité) et des bornes supérieures (des algorithmes avec véracité garantie) sur le meilleur rapport d'approximation qu'un algorithme avec véracité garantie peut avoir. Nous nous sommes également intéressés dans ce chapitre aux mécanismes de coordination avec véracité garantie et avons donné des bornes sur le meilleur rapport de coordination qu'ils peuvent avoir.

Dans le **chapitre 7**, nous avons étudié le routage de paquets autonomes et partant d'une même source dans un réseau "store and forward" qui est un chemin, un arbre, ou bien un anneau. Nous avons étudié la performance de mécanismes de coordination simples et naturels puisqu'ils sont basés sur la longueur de chaque paquet et/ou de la distance qu'il lui reste à parcourir pour arriver à sa destination. La performance de ces mécanismes a été étudiée pour les deux fonctions objectifs globales suivantes : la minimisation de la date d'arrivée moyenne des paquets, ainsi que la minimisation de la date d'arrivée du dernier paquet ; alors que le but de chaque paquet est d'arriver le plus rapidement à sa destination. Nous avons montré des différences significatives entre les performances de ces mécanismes, certains ayant un prix de l'anarchie qui est une petite constante, alors que d'autres ont un prix de l'anarchie non borné.

Remarques et perspectives générales

Nous avons étudié la performance de problèmes en présence de diverses difficultés. Tout d'abord l'approximation polynomiale nous a permis d'étudier le compromis nécessaire entre l'efficacité d'un algorithme et l'optimalité des solutions qu'il retourne. De même, le prix de l'anarchie a mesuré la perte de performance d'un algorithme en présence d'agents indépendants. Nous avons essayé d'associer à chaque résultat d'impossibilité la difficulté qui en est la cause, parmi les différentes contraintes que nous nous sommes fixées : avoir un algorithme polynomial, équitable, avec véracité garantie, qui retourne des solutions stables face à des agents autonomes, ou bien encore complètement distribué (un mécanisme de coordination). Par exemple, nous avons montré dans le chapitre 6 qu'il est possible, avec le modèle souple d'exécution des tâches, d'avoir un algorithme optimal et avec véracité garantie (mais en temps exponentiel) pour la date de fin de l'ordonnancement. Au contraire, avec le modèle fort d'exécution des tâches, il n'est pas possible d'avoir un algorithme avec véracité garantie optimal (ou même moins de $(3/2 - 1/2m)$-approché), et ce également en considérant les algorithmes s'exécutant en un temps exponentiel. De même, la NP-difficulté d'un problème est liée à une caractéristique précise du problème : le groupage de trafic dans un réseau optique en étoile est par exemple un problème NP-difficile seulement si le nombre de longueurs d'ondes est supérieur à 2 (voir chapitre 2) ; de la même façon, le voisinage présenté dans le chapitre 3 est explorable en temps polynomial si et seulement si les demandes des clients sont identiques, et ce quelque soit la matrice des distances considérée par exemple.

Les problèmes que nous avons étudiés sont tous liés aux réseaux. Bien sûr, il existe des différences sensibles entre les réseaux étudiés, et plus généralement entre deux réseaux quels qu'ils soient. Cependant, tout réseau semble pouvoir être modélisé à partir de graphes et d'ordonnancements, auxquels se rajoutent alors d'autres informations dans des réseaux plus complexes. Ces deux structures de données sont ainsi très étudiées en informatique théorique. Dans les chapitres 5 et 6 qui traitent des problèmes d'ordonnancements de tâches indépendantes, nous avons étudié le modèle CKN, dans lequel les tâches ne sont pas préemptées. Ce modèle avait été jusqu'à présent peu étudié dans le contexte de tâches indépendantes (et individualistes), le modèle de loin le plus étudié étant le modèle KP, dans lequel chaque tâche souhaite réduire la date de fin de sa machine. Dans le chapitre 7, nous avons étudié le prix de l'anarchie pour un problème de routage de paquets autonomes et individualistes dans un réseau "store and forward" – à la topologie certes particulière, ce réseau étant un arbre ou un anneau –, alors que le prix de l'anarchie concernant le routage de paquets autonomes et individualistes avait auparavant été étudié dans d'autres types de réseaux. Les réseaux "store and forward" occupent cependant une place prépondérante (internet est par exemple de ce type), et l'étude de l'impact de paquets détenus par des agents individualistes dans ce type de réseaux semble une perspective intéressante. Nous pourrons alors progressivement rajouter de nouvelles caractéristiques de ces réseaux afin d'atteindre un plus grand degré de réalisme.

Les problèmes d'optimisation face à des agents indépendants constituent un domaine émergent et en pleine expansion. En effet, les problèmes d'optimisation considérés jusqu'alors avaient souvent pour but d'améliorer les performances d'un même ordinateur, ou les performances au sein d'une entité gérée par un seul agent (ou des agents ayant des intérêts convergents). En présence d'un réseau comme internet, nous devons prendre en compte le fait que différents agents ont des intérêts divergents. Cette nouvelle contrainte entraîne de nouvelles méthodes de résolution des problèmes. Les algorithmes probabilistes semblent particulièrement adaptés à ce genre de situation, si bien sûr les contraintes du problème autorisent l'utilisation de tels algorithmes. Dans les problèmes étudiés dans les chapitres 5 et 6 les algorithmes probabilistes donnent par exemple de bien meilleurs résultats que les algorithmes déterministes. Ces algorithmes (ou politiques) probabilistes empêchent en effet les agents de connaître leur profit exact selon chacune des stratégies à leur disposition, et semblent donc les empêcher de manipuler le système à leur avantage.

Au fil de ces chapitres, nous avons remarqué que, dans un problème d'ordonnancement avec des tâches indépendantes et souhaitant minimiser leur date de fin, l'algorithme de liste SPT possède de bonnes propriétés. En effet, cet algorithme, en plus de minimiser la date de fin moyenne des tâches, produit de bons ordonnancements vis-à-vis de deux critères d'équité (voir chapitre 4), et est à véracité garantie (voir chapitre 6). Il peut de plus également être transformé facilement en mécanisme de coordination ayant les mêmes qualités que l'algorithme centralisé. Au contraire, l'algorithme de liste LPT n'est pas équitable (au sens des critères d'équité étudiés dans le chapitre 4) et n'est pas à véracité garantie, mais il possède un meilleur rapport d'approximation vis-à-vis de la date de fin de l'ordonnancement. Dans les chapitres 2 à 7 (et notamment dans les chapitres 5 à 7 avec des agents indépendants), la fonction objectif globale représentait le "bien-être" global de l'ensemble des protagonistes du problème. Dans le cas d'agents indépendants nous avons pourtant supposé que ces agents étaient égoïstes puisqu'ils se comportaient de façon à maximiser leur propre bien-être sans prendre en compte les conséquences que cela pouvait avoir sur le bien-être des autres agents. Cette contrainte peut induire un prix de l'anarchie important. Les protocoles maximisant le bien-être global des agents et ayant de plus de bonnes propriétés d'équité entre les agents pourraient être une perspective intéressante. En effet, il semble raisonnable de penser que les agents auront plus tendance à se comporter de manière égoïste si le protocole n'est pas "juste", par exemple s'il traite arbitrairement différemment deux agents ayant les mêmes caractéristiques (et ce même s'il maximise une fonction objectif globale représentant le bien-être général des agents). Les chapitres 4 (où l'on étudie un ordonnancement sous le point de vue de critères d'équité) et 5 (où l'on cherche à obtenir des ordonnancements dans lesquels on essaie de minimiser l'intérêt qu'auront des tâches individualistes à refuser l'ordonnancement proposé par le protocole) sont intéressants en ce sens. Bien que très subjective et liée au problème, la notion d'équité d'un protocole semble donc être également une perspective intéressante.

Bibliographie

[1] E. Aarts et J.K. Lenstra. *Local Search in Combinatorial Optimization*. Wiley, 1997.

[2] R.K. Ahuja, Ö. Ergun, J.B. Orlin et A.P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123:75–102, 2002.

[3] R.K. Ahuja, T.L. Magnanti et J.B. Orlin. *Network Flows : Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[4] P. Ambrosio et V. Auletta. Deterministic monotone algorithms for scheduling on related machines. *Dans Proceedings of the 2nd Workshop on Approximation and Online Algorithms (WAOA)*, volume 3351 de *LNCS*, pages 267–280. Springer, 2004.

[5] N. Andelman, Y. Azar et M. Sorani. Truthful approximation mechanisms for scheduling selfish related machines. *Dans Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 3404 de *LNCS*, pages 69–82. Springer, 2005.

[6] E. Angel, E. Bampis et F. Pascual. How fair are spt schedules. *Annals of Operation Research*, à paraître.

[7] E. Angel, E. Bampis et F. Pascual. Traffic grooming in a passive star wdm network. *Dans Proceedings of the 11th Colloquium on Structural Information and Communication Complexity (SIROCCO)*, volume 3104 de *LNCS*, pages 1–12. Springer, 2004.

[8] E. Angel, E. Bampis et F. Pascual. How fair are spt schedules. *Dans Proceedings of the 2nd Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA)*, pages 244–257, 2005.

[9] E. Angel, E. Bampis et F. Pascual. Propriétés des ordonnancements spt : approximation et critères de satisfaction. *Dans Actes du 6ème congrès de la société française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF)*, 2005.

[10] E. Angel, E. Bampis et F. Pascual. Truthful algorithms for scheduling selfish tasks. *Dans Proceedings of the 1st Workshop on Internet and Network Economics (WINE)*, volume 3828 de *LNCS*, pages 698–707. Springer, 2005.

[11] E. Angel, E. Bampis et F. Pascual. Mécanismes de coordination et routage dans les chemins, les arbres, et les anneaux. *Dans Actes des 8èmes rencontres francophones sur les aspects algorithmiques de télécommunications (ALGOTEL)*, pages 29–32, 2006.

[12] E. Angel, E. Bampis et F. Pascual. The price of approximate stability for scheduling tasks on two links. *Dans Proceedings of the 12th European Conference on Parallel Computing (Euro-Par)*, LNCS, pages 157–166. Springer, 2006.

[13] E. Angel, E. Bampis et F. Pascual. Truthful algorithms for scheduling selfish tasks. *Theoretical Computer Science*, 2006.

[14] E. Angel, E. Bampis, F. Pascual et J. Warnier. Couplage contraint et exploration efficace d'un voisinage exponentiel pour le problème de tournées de véhicules. *Dans Actes du 7ème congrès de la société française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF)*, 2006.

[15] E. Angel, E. Bampis, F. Pascual et J. Warnier. Restricted matchings and efficient exploration of an exponential neighborhood for the vehicle routing problem. Rapport technique IBISC RR 2006-01, IBISC, Université d'Evry Val d'Essonne, France, 2006.

[16] E. Angel, P. Christopoulos et V. Zissimopoulos. *Optimisation combinatoire 2 : concepts avancés*, chapitre Recherche locale : complexité et approximation. Hermès, 2005.

[17] E. Anshelevich, A. Dasgupta, J. Kleinberg, É. Tardos, T. Wexler et T. Roughgarden. The price of stability for network design with fair cost allocation. *Dans Proceedings of 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 295–304, 2004.

[18] A. Archer et E. Tardos. Truthful mechanisms for one-parameter agents. *Dans Proceedings of 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 482–491, 2001.

[19] F.M. auf der Heide et B. Vöcking. A packet routing protocol for arbitrary networks. *Dans Proceedings of the 12th Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 900 de *LNCS*, pages 291–302. Springer, 1995.

[20] V. Auletta, R. De Prisco, P. Penna et P. Persiano. Deterministic truthful approximation mechanisms for scheduling related machines. *Dans Proceedings of the 21st Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2996 de *LNCS*, pages 608–619. Springer, 2004.

[21] V. Auletta, P. Penna, R. De Prisco et P. Persiano. How to route and tax selfish unsplittable traffic. *Dans Proceedings of 16th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 196–204, 2004.

[22] V. Auletta, R. De Prisco, P. Penna et P. Persiano. The power of verification for one-parameter agents. *Dans Proceedings of the 31st International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 3142 de *LNCS*, pages 171–182. Springer, 2004.

[23] V. Auletta, R. De Prisco, P. Penna et P. Persiano. On designing truthful mechanisms for online scheduling. *Dans Proceedings of the 12th Colloquium on Structural Information and Communication Complexity (SIROCCO)*, volume 3499 de *LNCS*, pages 3–17. Springer, 2005.

[24] B. Awerbuch, Y. Azar et A. Epstein. The price of routing unsplittable flow. *Dans Proceedings of the 37th ACM Symposium on Theory of Computing (STOC)*, pages 57–66, 2005.

[25] Y. Azar et A. Epstein. The hardness of network design for unspittable flow wit selfish users. *Dans Proceedings of the 3rd Workshop on Approximation and Online Algorithms (WAOA)*, volume 3879 de *LNCS*, pages 41–54. Springer, 2005.

[26] P. Brucker. *Scheduling Algorithms*. Springer, 2001.

[27] J. Bruno, E.G. Coffman Jr. et R. Sethi. Algorithms for minimizing mean flow time. *Dans Proceedings of IFIP Congress*, pages 504–510. North-Holland, 1974.

[28] B. Chen, C.N. Potts et G.J. Woeginger. A review of machine scheduling : Complexity, algorithms and approximability. Rapport technique Woe-29, TU Graz, 1998.

[29] G. Christodoulou et E. Koutsoupias. The price of anarchy of finite congestion games. *Dans Proceedings of the 37th ACM Symposium on Theory of Computing (STOC)*, pages 67–73, 2005.

[30] G. Christodoulou, E. Koutsoupias et A. Nanavati. Coordination mechanisms. *Dans Proceedings of the 31st International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 3142 de *LNCS*, pages 345–357. Springer, 2004.

[31] E. Clarke. Multipart pricing of public goods. *Public Choices*, pages 17–33, 1971.

[32] R.K. Congram, C.N. Potts et S.L. Van de Velde. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14(1):52–67, 2002.

[33] R.W. Conway, W.L. Maxwell et L.W. Miller. *Theory of Scheduling*. Addison-Wesley, 1967.

[34] A. Czumaj et B. Vöcking. Tight bounds for worst-case equilibria. *Dans Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 413–420, 2002.

[35] V.G. Deineko et G.J. Woeginger. A study of exponential neighborhoods for the traveling salesman problem and the quadratic assignment problem. *Mathematical Programming, Ser. A*, 87:519–542, 2000.

[36] R. Dutta et G.N. Rouskas. Traffic grooming in wdm networks : Past and future. *IEEE Network*, 16(6):46–56, 2002.

[37] O. Ergun, J.B. Orlin et A.S-Feldman. Creating very large scale neighborhoods out of smaller ones by compounding moves : A study on the vehicle routing problem. *MIT Sloan Working Paper No 4393-02*, 2002.

[38] G. Finn et E. Horowitz. A linear time approximation algorithm for multiprocessor scheduling. *BIT*, 19:312–320, 1979.

[39] D. Fotakis, S. Kontogiannis, E. Koutsoupias, M. Mavronicolas et P. Spirakis. The structure and complexity of nash equilibria for a selfish routing game. *Dans Proceedings of the 29th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 2380 de *LNCS*, pages 123–134. Springer, 2002.

[40] M.R. Garey et D.S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness*. W. H. Freeman, San Francisco, 1979.

[41] R. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[42] R. Graham. Bounds on multiprocessor timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[43] T. Groves. Incentive in teams. *Econometrica*, 41(4):617–631, 1973.

[44] G. Gutin. Exponential neighbourhood local search for the traveling salesman problem. *Special issue of Computers and Operations Research on the traveling salesman problem*, 26:313–320, 1999.

[45] G. Gutin, A. Yeo et A. Zverovitch. Exponential neighborhoods and domination analysis for the TSP. *Dans* G. Gutin et A.P. Punnen, éditeurs. *Traveling salesman problem and its variations*, volume 12 de *Combinatorial Optimization*, pages 223–256. Kluwer, 2002.

[46] M. T. Hajiaghayi, R. D. Kleinberg, M. Mahdian et D. C. Parkes. Online auctions with re-usable goods. *Dans Proceedings of the 6th ACM Conference on Electronic Commerce (EC)*, pages 165–174, 2005.

[47] A. Hall, S. Hippler et M. Skutella. Multicommodity flows over time : Efficient algorithms and complexity. *Proceedings of the 30th International Colloquium on Automata, Languages, and Programming (ICALP) (à paraître dans Theoretical Computer Science)*, 2003.

[48] D.S. Hochbaum, éditeur. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997.

[49] S. Huang, R. Dutta et G.N. Rouskas. Traffic grooming in path, star, and tree networks : Complexity, bounds, and algorithms. *Dans Proc of ACM SIGMETRICS (et à paraître dans IEEE Journal on selected areas in communications)*, pages 298–299, 2003.

[50] J. Hurink. An exponential neighborhood for a one-machine batching problem. *OR Spectrum*, 21(4):461–476, 1999.

[51] N. Immorlica, L. Li, V.S. Mirrokni et A. Schulz. Coordination mechanisms for selfish scheduling. *Dans Proceedings of the 1st Workshop on Internet and Network Economics (WINE)*, volume 3828 de *LNCS*, pages 55–69. Springer, 2005.

[52] A. Itai, M. Rodeh et S.L. Tanimoto. Some matching problems for bipartite graphs. *Journal of the ACM*, 25:517–525, 1978.

[53] D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, 1974.

[54] E. Koutsoupias, M. Mavronicolas et P. Spirakis. Approximate equilibria and ball fusion. *Theory of Computing Systems*, 36(6):683–693, 2003.

[55] E. Koutsoupias et C.H. Papadimitriou. Worst-case equilibria. *Dans Proceedings of the 16th Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1563 de *LNCS*, pages 404–413. Springer, 1999.

[56] A. Kovacs. Fast monotone 3-approximation algorithm for scheduling related machines. *Dans Proceedings of the 13th European Symposium on Algorithms (ESA)*, volume 3669 de *LNCS*, pages 616–627. Springer, 2005.

[57] A. Kumar et J. Kleinberg. Fairness measures for ressource allocation. *Dans Proceedings of 41st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 75–85, 2000.

[58] F.T. Leighton, B.M. Maggs et S.B. Rao. Packet routing and job-shop scheduling in o(congestion + dilatation) steps. *Combinatorica (preliminary version in FOCS 1988)*, pages 167–186, 1994.

[59] F.T. Leighton, B.M. Maggs et A.W. Richa. Fast algorithms for finding o(congestion + dilatation) packet routing schedules. *Combinatorica*, pages 375–401, 1999.

[60] R. Lipton, E. Markakis et A. Mehta. Playing large gamed using simple strategies. *Dans Proceedings of the 4th ACM Conference on Electronic Commerce (EC)*, pages 36–41, 2003.

[61] E. Modiano et P.J. Lin. Traffic grooming in wdm networks. *IEEE Communications*, 39(7): 124–129, 2001.

[62] O. Morgenstern et J. Von Neumann. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[63] J.F. Nash. Non-cooperative games. *Annals of Mathematics*, 54:286–295, 1951.

[64] N. Nisan et A. Ronen. Algorithmic mechanism design. *Dans Proceedings of the 31st ACM Symposium on Theory of Computing (STOC)*, pages 129–140, 1999.

[65] M.J. Osborne et A. Rubinstein. *A course in game theory*. The MIT Press, 1994.

[66] R. Ostrovsky et Y. Rabani. Universal o(congestion + dilatation + $\log^{1+\varepsilon} n$) local control packet switching algorithms. *Dans Proceedings of the 29th ACM Symposium on Theory of Computing (STOC)*, pages 644–653, 1997.

[67] C.H. Papadimitriou. Algorithms, games, and the internet. *Dans Proceedings of the 33rd ACM Symposium on Theory of Computing (STOC)*, pages 749–753, 2001.

[68] C. Phillips, C. Stein et J. Wein. Scheduling jobs that arrive over time. *Dans Proceedings of the Fourth Workshop on Algorithms and Data Structures (WADS)*, volume 955 de *LNCS*, pages 86–97. Springer, 1995.

[69] R. Porter. Mechanism design for online real-time scheduling. *Dans Proceedings of the 5th ACM Conference on Electronic Commerce (EC)*, pages 61–70, 2004.

[70] S. Rajagopalan et V.V. Vazirani. Primal-dual rnc approximation algorithms for set cover and covering integer programs. *SIAM Journal on Computing*, 28:526–541, 1999.

[71] R. W. Rosenthal. A class of games possessing pure strategy nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.

[72] T. Roughgarden et E. Tardos. How bad is selfish routing ? *Journal of the ACM*, 49(2):236–259, 2002.

[73] A. S. Schultz et N. Stier Moses. On the performance of user equilibria in traffic networks. *Dans Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 86–87, 2003.

[74] B. Shepherd et A. Vetta. The demand matching problem. *Dans Proceedings of the 9th Conference on Integer Programming and Combinatorial Optimisation (IPCO)*, volume 2337 de *LNCS*, pages 457–474. Springer, 2002.

[75] D.B. Shmoys, C. Stein et J. Wein. Improved approximation algorithms for shop scheduling problems. *Dans Proceedings of the 2nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 148–15, 1991.

[76] C. Stein et J. Wein. On the existence of schedules that are near-optimal for both makespan and total weighted completion time. *Operations Research Letters*, 21(3):115–122, 1997.

[77] P.M. Thompson et H.N. Psaraftis. Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Operations Research*, 41:935–946, 1993.

[78] P. Toth et D. Vigo. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematic, 2001.

[79] V. Vazirani. *Approximation algorithms*. Springer, 2001.

[80] W. Vickrey. Counterspeculation, auctions and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

[81] J. Xu et J.P. Kelly. A network flow-based tabu search heuristic for the vehicle routing problem. *Transportation Science*, 30:379–393, 1996.

[82] K. Zhu et B. Mukherjee. A review of traffic grooming in wdm optical networks : Architecture and challenges. *Optical Networks Magazine*, 4(2):55–64, 2003.

ANNEXE

Résultats de la section 6.2 dans le cas où les machines ne sont plus identiques mais ont des vitesses d'exécution différentes

Théorème A1 *Supposons que nous avons un nombre entier m de machines $P_1, \ldots, P_m$ telles que la vitesse de la machine P_i est v_i, $v_1 = 1$, et $v_1 \leq \cdots \leq v_m$. Il n'existe pas d'algorithme déterministe avec véracité garantie ayant un rapport d'approximation inférieur à $2 - \frac{v_m}{\sum_{i=1}^m v_i}$.*

Preuve : Considérons l'instance suivante : n tâches de longueurs 1, avec n un nombre entier tel que $n >> \sum_{i=1}^m v_i$. Supposons que nous avons un algorithme déterministe avec véracité garantie $\mathcal{A}$ ayant un rapport d'approximation égal à ρ. Nous allons montrer que ρ est nécessairement supérieur ou égal à $2 - \frac{v_m}{\sum_{i=1}^m v_i}$). Soit t la tâche qui a la date de fin maximum, C_t, dans l'ordonnancement retourné par $\mathcal{A}$. Nous savons que $C_t \geq \frac{n}{\sum_{i=1}^m v_i}$. En effet, s'il est possible d'affecter $\frac{n\, v_j}{\sum_{i=1}^m v_i}$ tâches à chaque machine P_j, avec $j \in \{1, \ldots, m\}$, alors la date de fin de chaque machine sera égale à $\frac{n}{\sum_{i=1}^m v_i}$. Si, pour un certain $j \in \{1, \ldots, m\}$, il n'est pas possible d'affecter $\frac{n\, v_j}{\sum_{i=1}^m v_i}$ tâches à la machine P_j (car $\frac{n\, v_j}{\sum_{i=1}^m v_i}$ n'est pas un nombre entier), alors il existe une machine P_k sur laquelle il y a plus de $\frac{n\, v_k}{\sum_{i=1}^m v_i}$ tâches, et la date de fin de la dernière tâche de cette machine est donc supérieure à $\frac{n}{\sum_{i=1}^m v_i}$.

Supposons que la tâche t déclare $l'_t = \frac{(n-1)\, v_m}{\sum_{i=1}^{m-1} v_i}$ au lieu de 1. Soit OPT la date de fin d'un ordonnancement optimal dans lequel il y a $n-1$ tâches de longueur 1 et une tâche de longueur l'_t. Nous avons : $\frac{(n-1)}{\sum_{i=1}^{m-1} v_i} \leq OPT < \frac{(n-1)}{\sum_{i=1}^{m-1} v_i} + 1$. En effet, $\frac{(n-1)}{\sum_{i=1}^{m-1} v_i}$ est la date à laquelle toutes les machines finissent d'exécuter leurs tâches en même temps, si cela est possible : dans ce cas la tâche de longueur l'_t est sur P_m et il y a $\frac{(n-1)\, v_j}{\sum_{i=1}^{m-1} v_i}$ tâches de longueur 1 sur la machine P_j, pour chaque $j \in \{1, \ldots, m-1\}$. S'il n'est pas possible que toutes les machines se terminent en même temps, alors nous ordonnançons (au plus) $\lceil \frac{(n-1)\, v_j}{\sum_{i=1}^{m-1} v_i} \rceil$ tâches de longueur 1 sur la machine P_j, pour tout $j \in \{1, \ldots, m-1\}$: toutes les tâches sont alors ordonnancées, et la date de fin de chaque machine P_i est inférieure à $\frac{(n-1)}{\sum_{i=1}^{m-1} v_i} + \frac{1}{v_i} \leq \frac{(n-1)}{\sum_{i=1}^{m-1} v_i} + 1$.

L'algorithme $\mathcal{A}$ n'est pas à véracité garantie si la date de fin de t qui déclare l'_t est inférieure à $\frac{n}{\sum_{i=1}^m v_i}$, puisque la date de fin de t quand elle déclare 1 est supérieure ou égale à $\frac{n}{\sum_{i=1}^m v_i}$. Soit S_A l'ordonnancement retourné par l'algorithme $\mathcal{A}$ quand il doit ordonnancer $(n-1)$ tâches de longueur 1 et une tâche de longueur l'_t. L'algorithme $\mathcal{A}$ n'est pas à véracité garantie si au moins une unité de la tâche de longueur l'_t se termine avant $\frac{n}{\sum_{i=1}^m v_i}$ unités de temps dans S_A. Ainsi, si nous voulons que $\mathcal{A}$ soit à véracité garantie, alors moins d'une unité de la tâche de longueur l'_t doit se terminer avant la date $\frac{n}{\sum_{i=1}^m v_i}$ dans S_A. Dans ce cas là, la date de fin de S_A est supérieure ou égale à $\frac{n}{\sum_{i=1}^m v_i} + (\frac{n-1}{\sum_{i=1}^{m-1} v_i} - \frac{1}{v_m})$, où $\frac{n-1}{\sum_{i=1}^{m-1} v_i} - \frac{1}{v_m}$ est la durée minimum nécessaire pour exécuter une tâche de longueur $l'_t - 1$. Ainsi, si $\mathcal{A}$ est à véracité garantie, la date de fin de S_A est supérieure ou égale à $\frac{n}{\sum_{i=1}^m v_i} + (\frac{n-1}{\sum_{i=1}^{m-1} v_i} - \frac{1}{v_m}) \geq \left(\frac{\sum_{i=1}^{m-1} v_i}{n-1+\sum_{i=1}^{m-1} v_i}\right)\left(\frac{n}{\sum_{i=1}^m v_i} + \frac{n-1}{\sum_{i=1}^{m-1} v_i} - \frac{1}{v_m}\right) OPT$, car nous avons vu que $OPT \leq \frac{(n-1)}{\sum_{i=1}^{m-1} v_i} + 1$. Cette dernière expression est égale à $\frac{n-1}{n-1+\sum_{i=1}^{m-1} v_i} + \left(\frac{n}{n-1+\sum_{i=1}^{m-1} v_i}\right) \frac{\sum_{i=1}^{m-1} v_i}{\sum_{i=1}^m v_i} - \frac{\sum_{i=1}^{m-1} v_i}{v_m\, (n-1+\sum_{i=1}^{m-1} v_i)}$), et tend vers $1 + \frac{\sum_{i=1}^{m-1} v_i}{\sum_{i=1}^m v_i} = 2 - \frac{v_m}{\sum_{i=1}^m v_i}$, quand $n >> \sum_{i=1}^m v_i$.

Ainsi, si $\rho < 2 - \frac{v_m}{\sum_{i=1}^m v_i}$, alors il existe une instance de n tâches de longueur 1 dans laquelle une des tâches aura intérêt à déclarer l'_t à la place de 1, sa vraie longueur. Il n'existe donc pas d'algorithme déterministe avec véracité garantie ayant un rapport d'approximation inférieur à $2 - \frac{v_m}{\sum_{i=1}^m v_i}$.
□

Le théorème 6.1 est donc un corollaire de ce théorème quand toutes les machines ont des vitesses identiques.
Nous pouvons adapter de la même manière la preuve du théorème 6.2, pour montrer le théorème suivant dans le cas de machines avec des vitesses différentes :

Théorème A2 *Supposons que nous avons un nombre entier m de machines $P_1, \ldots, P_m$ telles que la vitesse de la machine P_i est v_i, $v_1 = 1$, et $v_1 \leq \cdots \leq v_m$. Il n'existe pas d'algorithme probabiliste avec véracité garantie ayant un rapport d'approximation inférieur à $2 - \frac{v_m}{\sum_{i=1}^m v_i}$.*

Le théorème 6.2 est donc un corollaire de ce théorème quand toutes les machines ont des vitesses identiques.