Tomographie discrète, calcul quantique et ordonnancement

Christoph Dürr

Tomographie discrète, calcul quantique et ordonnancement

2005

Christoph Dürr

mémoire d'habilitation

Table des matières

Chapitre 1
Introduction

Les trois faces d'une médaille

Trois domaines de l'algorithmique, tomographie discrète, calcul quantique, ordonnancement. Ces domaines sont très différents de par leurs motivations, leur techniques, et leur communauté scientifique, à première vue... à deuxième vue aussi. J'ai été surpris cependant d'observer certaines similitudes. Par exemple, l'échange élémentaire, qui est une opération importante sur les matrices en tomographie discrète se retrouve en ordonnancement quand on veut montrer qu'un programme linéaire dont les variables s'organisent en matrice, admettent toujours une solution entière. En revanche mes contributions dans ces domaines sont souvent des réductions entre des problèmes. S'il s'agit d'établir la NP-complétude, ou de montrer qu'un problème peut être résolu en temps polynomial en le réduisant à un cas faisable, ces résultats tissent des liens entre les problèmes. J'ai un peu l'impression que c'est l'essence du problème qui s'en dégage, le pourquoi de sa difficulté ou de sa facilité. Finalement ce sont les techniques développées qui sont importantes, et qui, je l'espère, peuvent servir un jour à résoudre d'autres problèmes.

Ce document est alors divisé en trois parties, un par domaine auquel j'ai contribué. Dans chaque partie après avoir exposé les motivations et principaux résultats, un tableau montre le paysage de différents problèmes, en marquant d'un fond gris les principales contributions personnelles. Ensuite celles-ci sont décrites en mettant l'accent sur les techniques employées. De temps en temps, un résultat est développé à titre d'exemple. Pour tous les détails, et pour les résultats plus compliqués, je renvoie aux articles en question. J'ai choisi des problèmes importants dans les domaines, qui sont restés ouverts en dépit de nos efforts. Ces problèmes sont décrits dans les sections concernées, et en tant que drapeaux plantés sur la plage de notre île de connaissance, ils invitent le lecteur à les résoudre. (Quelle phrase pathétique !)

Finalement il y a aussi un travail effectué en collaboration avec Ivan Rapaport et Guillaume Theyssier [40] qui n'a pas trouvé sa place parmi ces trois chapitres. Il s'agit d'une étude des automates cellulaires à une dimension par le regard de la complexité de communication. Longtemps j'ai eu un rapport difficile avec ce travail, qui comporte beaucoup de résultats expérimentaux, mais avec le recul, je l'apprécie beaucoup. Cependant il ne pouvait pas faire l'objet d'une section à part entière.

Les chapitres suivants sont donc indépendants et peuvent être lus dans un ordre arbitraire, l'ordre historique a été choisi ici. Une dernière remarque : pratiquement tous les algorithmes développés ont été implémentés et peuvent être exécutés directement à partir du web, afin d'obtenir une intuition sur le problème et l'algorithme en question.

1.1 Remerciements

Je tiens à remercier pour la relecture de ce texte, ma mère et mes collègues Pierre Fraigniaud et Sophie Laplante. J'ai une pensée également pour tous mes co-auteurs qui ont contribué à ce travail, en particulier Marek Chrobak qui m'a beaucoup appris. Et finalement je remercie chaleureusement ma femme pour son soutien et sa patience.

Chapitre 2

Tomographie discrète

Ce chapitre est basé sur les articles personnels suivants :

1. [34] *Reconstructing Polyatomic Structures from Discrete X-Rays: NP-Completeness Proof for Three Atoms*, avec Marek Chrobak. Theoretical Computer Science, vol 259, pp. 81-98, 2001. Version journal de Proc. of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS), LNCS vol 1450, pp. 185–193, 1998.

2. [33] *Reconstructing hv-Convex Polyominoes from Orthogonal Projections*, avec Marek Chrobak. Information Processing Letters, vol. 69, pp. 283–289, 1999.

3. [38] *Tiling with bars under tomographic constraints*, avec Eric Goles, Ivan Rapaport et Eric Rémila. Theoretical Computer Science, vol. 290, pp. 1317–1329, 2003.

4. [32] *A Note on Tiling under Tomographic Constraints*, avec Marek Chrobak, Peter Couperus et Gerhard Woeginger. Theoretical Computer Science, vol. 290, pp. 2125–2136, 2003.

2.1 Préliminaires

Par *tomographie* on entend la reconstruction d'objets à partir de leurs projections. Par exemple en imagerie médicale,, on construit l'image en 3 dimensions du cerveau d'un patient à partir de photos par rayons X du crâne, prises de différents angles. Dans le cadre du contrôle non-destructif industriel, on veut détecter des bulles d'air dans un bloc en fonte. Ces applications forment ce qu'on appelle la tomographie *continue,* domaine de recherche bien décrit dans un livre de Richard Gardner [46].

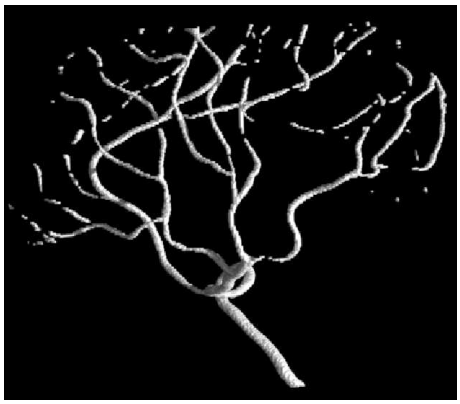


Figure 2.1. reconstruction d'un branchement d'artères à partir de deux projections discrètes de 0 et 90 degrés (source Stefan Weber, Université de Mannheim).

Depuis la fin des années 80 on s'intéresse aux versions discrètes de ces problèmes de reconstruction. Ici les projections sont discrètes, peu nombreuses et l'objet à reconstruire est un objet discret, comme par exemple un ensemble de cellules dans une grille. Ce domaine de recherche est alors appelé tomographie *discrète.* Il est motivé en partie par une technique appelée QUANTITEM [53, 70], similaire à un microscope à électrons et qui permet de compter le nombre d'atomes dans chaque colonne d'une structure métallique où les atomes sont alignés en forme de grille. Alors que les algorithmes de reconstruction en tomographie continue sont basés essentiellement sur des transformations de Radon inverse, ils ne peuvent pas être appliqués à la tomographie discrète. Il faut alors trouver de nouvelles techniques d'algorithmique combinatoire. Le passage au discret rend certains problèmes de tomographie difficiles, et suivant les problèmes, les techniques utilisées pour les résoudre sont parfois similaires, mais parfois aussi complètement différentes.

Ce chapitre ne se veut pas une compilation de tous les résultats de ce domaine, pour cela je renvoi à une collection d'articles [50]. Je vais plutôt exposer les familles de problèmes dans lesquelles je suis intervenu. Le problème de base consiste à reconstruire une matrice 01 en respectant les *projections,* qui sont des sommes données sur les lignes et les colonnes. Ces matrices codent des ensembles de cellules d'une grille. Comme la solution est souvent loin d'être unique, on considère le problème de reconstruire un ensemble de cellules aux propriétés particulières, comme celle d'être convexe. Ce problème n'a pas encore été résolu, c'est pourquoi on s'intéresse à une propriété plus faible, qui constitue les polyominos hv-convexes. Avec Marek Chrobak j'ai amélioré un algorithme de reconstruction de ces ensembles de cellules. Nous avons aussi travaillé sur la reconstruction de placements de jetons colorés sur une grille, où nous avons amélioré un résultat de NP-complétude. Un des nombreux problèmes ouverts en tomographie discrète est la reconstruction de pavages par des dominos. Avec Eric Goles, Ivan Rapaport et Eric Rémila nous avons obtenu des résultats sur des cas particuliers et des cas plus généraux, qui sont exposés en section ?. Puis finalement dans un souci d'unification, j'ai consi-

déré le problème de reconstruction de pavages en général, et tenté de classifier sa complexité en fonction de l'ensemble des tuiles considéré. Ce dernier travail a été effectué avec Peter Couperus, Marek Chrobak et Gerhard Woeginger. Mais avant de détailler nous contributions, place à une introduction à la tomographie discrète à l'aide d'un problème concret très simple.

2.1.1 Reconstruction de matrices 0-1

Le problème suivant a été étudié dans les années 60, et la solution est très utile pour les autres problèmes de tomographie discrète.

PROBLÈME DE RECONSTRUCTION DE MATRICES 0-1

Entrée. $2n$ entiers $r_1, ..., r_n, c_1, ..., c_n$ pour $n \geqslant 1$

Sortie. une matrice $M \in \{0,1\}^{n \times n}$ telle que pour tout $1 \leq i, j \leq n$

$$r_i = \sum_{k=1}^{n} M_{\mathrm{ik}} \text{ et } c_j = \sum_{k=1}^{n} M_{\mathrm{kj}}.$$

Clairement la solution n'est pas forcément unique si elle existe, par exemple l'instance $1, ..., 1$ a $n!$ solutions, les matrices de permutations. Ce problème peut aussi être vu comme un problème de graphe, M est alors la matrice d'adjacence d'un graphe bi-parti avec des sommets aux degrés spécifiés par $r_1, ..., r_n, c_1, ..., c_n$.

Ce problème fut étudié en 1963 par Ryser qui donna une caractérisation exacte des projections admettant une solution. Ce problème étant important nous allons reproduire ce résultat. Pour cela il nous faut deux définitions, le *dual* et la *dominance* :

Soit $v \in \mathbb{N}^n$ un vecteur trié ($v_i \geq v_{i+1}$ pour tout $1 \leq i \leq n-1$) et satisfaisant $0 \leq v_i \leq n$ pour tout $1 \leq i \leq n$. Graphiquement on peut représenter un tel vecteur comme un tableau de n barres horizontaux, alignés sur leur gauche, et la i-ème barre est de longueur v_i, voir figure 2.2. Ce tableau peut aussi être vu comme l'union de n barres verticales, alignées sur leur haut. La hauteur de la j-ème barre est alors $\bar{v}_j := |\{i : v_i \leq j\}|$. Le vecteur $\bar{v}$ est de nouveau un vecteur trié, et d'entrées comprises entre 0 et n. Notons que si $u = \bar{v}$ alors $\bar{u} = v$, ceci motive le terme "*dual*".

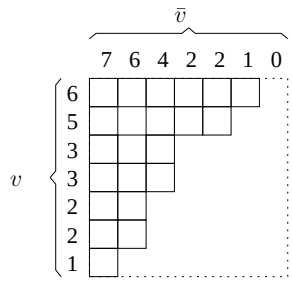


Figure 2.2. Le dual d'une projection.

On définit un ordre partiel sur les vecteurs de même dimension, appelé *ordre de dominance*. Soient les vecteurs $u, v \in \mathbb{N}^n$. On dit que u domine v, noté $v \preceq u$, si pour tout $1 \leq i \leq n$ on a $\sum_{j=1}^{i} v_j \leq \sum_{j=1}^{i} u_j$.

Cet ordre a la jolie propriété $u \preceq v$ si et seulement si $\bar{v} \preceq \bar{u}$. Pour la preuve je renvoi à l'excellent article de Brualdi [26].

Proposition 2.1. *Il existe une matrice 0-1 aux projections r, c si et seulement si $r \preceq \bar{c}$ (ou de manière équivalente $\bar{r} \preceq c$).*

Cette propriété est vérifiable en temps $O(n)$, il faut commencer par trier les vecteurs en temps $O(n)$ avec le tri par paniers (*bucket sort*), puis effectuer les n tests d'inégalité en temps linéaire. Une solution au problème peut aussi facilement être produite, par un algorithme *glouton*. Il suffit de procéder ligne par ligne, disons des indices n à 1 et pour la i-ème ligne placer les r_i '1's dans les colonnes j dont la projection c_j est maximale. Ensuite on décrémente les projections c_j de ces colonnes et on recommence. On peut facilement vérifier que ceci préserve l'ordre de dominance $r \preceq \bar{c}$.

Ce problème a également beaucoup de structure. Soit $\mathcal{M}$ l'ensemble des matrices qui sont solution au problème pour des projections (r, c) fixées. Alors $\mathcal{M}$ est connecté par l'opération élémentaire suivante. Pour une matrice $M \in \mathcal{M}$ soient les indices i_1, i_2, j_1, j_2. Si $M_{i1j1} = M_{i2j2} = 1$ et $M_{i1j2} = M_{i2j1} = 0$, alors l'échange de 0 et 1 dans ces quatre variables préserve les projections et résulte en une autre matrice $M' \in \mathcal{M}$. On peut obtenir n'importe quelle matrice de $\mathcal{M}$ par M avec simplement des opérations élémentaires. C'est dans ce sens que $\mathcal{M}$ est connecté.

2.2 Contributions

2.2.1 Reconstruction de polyominos hv-convexes

Nous avons vu que parfois la solution au problème de reconstruction de matrices 0-1 n'est pas unique. Si on a des informations supplémentaires sur l'objet à reconstruire, on devrait s'en servir pour restreindre l'espace des solutions. Dans la tomographie médicale par exemple on veut retrouver la forme du coeur d'un patient à l'aide de très peu de projections, car les rayons X sont nuisibles à la longue, et que chaque mesure prend beaucoup de temps. Le problème abstrait associé consiste alors à reconstruire une matrice 0-1, qui est la matrice caractéristique d'un ensemble connexe de cellules de la grille $n \times n$. Ce problème est encore ouvert à ce jour. On considère alors une relaxation, à savoir la reconstruction de *polyominos hv-convexes*. Elena Barcucci, Alberto del Lungo, Maurice Nivat et Renzo Pinzani ont donné un algorithme polynomial pour ce problème [16], que nous allons maintenant définir formellement.

Définition 2.2. *Un ensemble de cellules est h-convexe, si pour toute colonne j il existe un intervalle de lignes I (éventuellement vide) tel que toute cellule (i, j) fait partie de l'ensemble si et seulement si $i \in I$. La notion de v-convexité est définie de la même manière pour les lignes. Un ensemble est connexe (polyomino) si le graphe sur l'ensemble des cellules est connexe, où deux cellules sont reliées par une arête si elles sont adjacentes dans une même colonne ou ligne.*

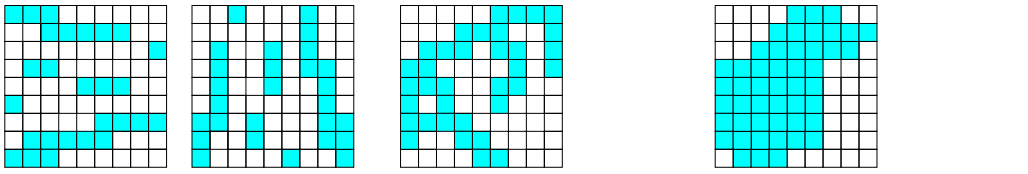


Figure 2.3. Propriétés d'ensembles de cellules.

Barcucci et al ont alors obtenu ces résultats intéressants. Reconstruire une matrice 0-1 hv-convexe et connexe peut être résolu en temps polynomial, comme l'est aussi le problème de reconstruire une matrice 0-1 sans conditions sur la solution. Par contre le même problème avec des conditions partielles sur la convexité, connexité est NP-complet au sens fort (voir tableau).

	-	polyomino
-	polynomial [67]	NP-complet [75]
h-convexe	NP-complet [15]	NP-complet [15]
hv-convexe	NP-complet [75]	polynomial [15] puis [33]

Tableau 2.1. (par Barcucci et al. [15]) Complexité du problème de reconstruction pour plusieurs conditions sur l'ensemble à reconstruire. Notre contribution est marquée en gris.

Leur algorithme est la version discrète d'un algorithme de la tomographie continue, où on veut reconstruire un polygone convexe [55]. L'algorithme construisait itérativement deux ensembles de cellules, une *enveloppe* S et un *noyau* C, avec la propriété que toutes les solutions M satisfont $C \subseteq M \subseteq S$. Quand il n'est plus possible de réduire S ou d'augmenter C une formule 2-SAT est générée où les variables représentent les cellules $S \backslash C$. Il y a alors une bijection entre les assignations satisfaisantes et les solutions au problème de reconstruction. Ces deux parties de l'algorithme donnaient un temps de calcul en $O(n^8)$ au pire des

cas. On est très tenté de résoudre le problème directement via une formule logique dont chaque variable booléenne est associée à une cellule de la grille. Le problème est qu'on obtient alors une formule 3-SAT. Notre idée avec Marek Chrobak était d'associer quatre variables à toute cellule, une pour chacune des 4 régions en dehors de M délimitées par la grille carrée [33]. Pour être précis la formule entière est encore une formule 3-SAT. Mais elle est la conjonction de $O(n^2)$ formules 2-SAT de taille $O(n^2)$ chacune, qui peuvent être vérifiées individuellement.

Les $O(n^2)$ formules 2-SAT ne diffèrent que dans un petit nombre de clauses. Plutôt que de résoudre chaque formule individuellement, il serait alors intéressant d'avoir un algorithme en-ligne pour 2-SAT.

> 2-SAT EN-LIGNE : trouver une structure de donnée dynamique qui stocke une formule 2-SAT et qui admet les opérations ajoute(clause), enlève(clause), résout_la_formule(). Le coût amorti devrait être $o(n)$.

L'algorithme de résolution d'une formule 2-SAT [8], construit tout simplement le graphe dirigé sur les littéraux, où chaque clause $x \vee y$ génère les arcs $\overline{x} \Rightarrow y$ et $\overline{y} \Rightarrow x$. Puis il calcule subtilement en $O(m)$ opérations les composantes fortement connexes, et vérifie qu'il n'y pas une variable qui se trouve dans la même composante que sa négation. Il y a des algorithmes en-ligne pour la connexité d'un graphe, voir un travail de Valerie King et Monica Henzinger [52]. Mais pour les graphes dirigés le problème est beaucoup plus difficile, en particulier en ce qui concerne l'opération de suppression d'arcs.

Notre article sur la reconstruction de polyominos hv-convexes [33], donnait aussi un algorithme en temps linéaire du cas particulier, où il existe une ligne i avec la projection $r_i = n$. Dans ce cas, toute solution contient la ligne i entièrement, ce qui structure fortement le polyomino. J'ai été contacté par Maurice Nivat, qui me faisait remarquer qu'ils savaient déjà depuis longtemps que ce cas est polynomial, c'est pourquoi je ne le décris pas ici. D'ailleurs leur approche est bien plus élégante et repose sur un problème de permutations.

2.2.2 Reconstruction d'une matrice colorée

Une généralisation possible de ce problème est la suivante. Décrivons d'abord le problème initial différemment. Maintenant il s'agit de placer des jetons sur la matrice, à raison d'au plus un jeton par case, en respectant des contraintes données sur le nombre de jetons dans chaque colonne et chaque ligne. La généralisation consiste à colorier les jetons. A présent pour un ensemble de couleurs fixé, les contraintes existent pour chaque couleur et chaque ligne/colonne. Formellement les projections sont des nombres r_{ai} et c_{aj} pour chaque couleur a, chaque ligne i et colonne j.

Reconstruction de Placement de Jetons Colorés pour k Couleurs

Entrée. des entiers r_{ai}, c_{aj} pour chaque couleur $a \in \{1,...,k\}$ et $i,j \in \{1,...,n\}$

Sortie. une matrice $M \in \{0,1,...,k\}^{n\times n}$ tel que pour tout $a \in \{1,...,k\}$ et i, $j \in \{1,...,n\}$

$$r_{ai} = |\{j : M_{ij} = a\}| \text{ and } c_{aj} = |\{i : M_{ij} = a\}|.$$

On retrouve alors le problème initial pour $k=1$. La question était posée de savoir si ce problème est polynomial pour un nombre de couleurs plus grand que 1. Gardner et al.[47] y ont répondu en donnant une preuve de NP-complétude pour $k \geq 6$, laissant ouvert les cas d'un nombre de couleurs entre 2 et 5. Nous avons d'abord simplifié leur preuve et construit une réduction du problème de Vertex-Cover qui utilisait comme gadget un bloc dans lequel la présence ou l'absence d'un jeton vert dans certaines cases codaient un ensemble de sommets : Dans la réduction 3 couleurs sont utilisées pour forcer une structure, illustrée en figure 2.4, qui est composée de blocs le long et juste au-dessus de la diagonale. Il y a aussi une bande aux deux extrémités de la diagonale. Les projections sont telles qu'un placement de jetons verts dans la première bande, force le placement dans le premier bloc de la diagonale qui se trouve sur les mêmes lignes. Celui-ci, à son tour, force le placement des jetons verts dans le premier bloc au dessus de la diagonale et ainsi de suite. Ces blocs agissent ainsi comme des miroirs, dupliquant à l'identique la configuration des jetons verts sur tous les blocs au-dessus de la diagonale. Une cinquième couleur est ensuite utilisée pour vérifier que l'ensemble des sommets codé par la configuration couvre bien toutes les arêtes.

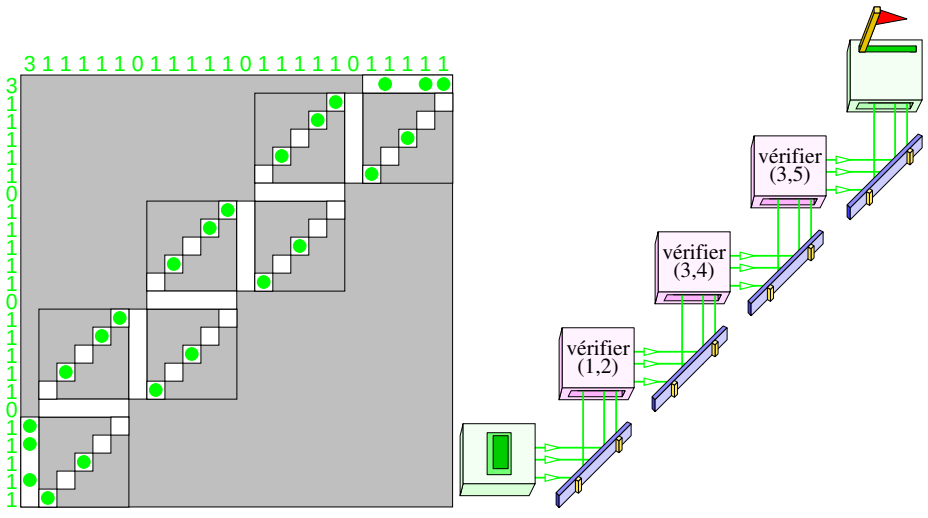


Figure 2.4. Idée de la réduction de Gardner et Gritzman et aussi de la nôtre.

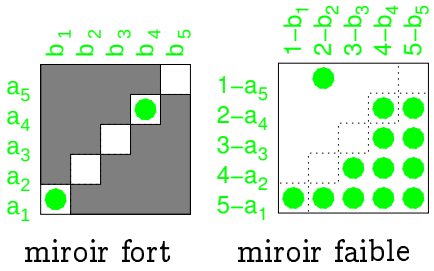


Figure 2.5. Gadget simple, le miroir fort admet une solution si et seulement si les deux vecteurs binaires a, b satisfont $a = b$. Tandis que le miroir faible admet une solution si et seulement si $a \preceq b$.

Notre contribution était de se rendre compte que des miroirs faibles sont suffisants, ce sont des miroirs qui assurent que dans une solution les projections sur les lignes sont dominées par celles sur les colonnes. Il fallait ensuite utiliser une propriété sur la profondeur de l'ordre partiel de dominance. Notre construction et sa preuve ne sont pas faciles à décrire, nous renvoyons à l'article pour plus de détails [34]. Cette nouvelle preuve montre que le problème est NP-complet pour $k \geq 3$, laissant ouvert le cas $k = 2$, ce qui est assez agaçant, je dois l'avouer.

2.2.3 Reconstruction de pavages de dominos

Un autre problème ouvert, tout aussi simple, est la reconstruction de pavages de dominos. Considérons le graphe d'adjacence d'une grille, dont les sommets sont les cellules et il y a une arête entre deux sommets si les cellules correspondantes sont adjacentes dans une même colonne ou ligne. Alors un pavage de dominos de la grille correspond par définition à un couplage parfait dans le graphe associé. Le problème consiste alors à construire un pavage de dominos d'une grille en respectant des contraintes données sur le nombre de dominos verticaux et horizontaux dans chaque ligne et chaque colonne. Il y a plusieurs manières équivalentes de définir ces contraintes, nous allons utiliser la suivante. Dans chaque tuile exactement une cellule est marquée, et les projections comptent le nombre de marques de chaque tuile sur les lignes et colonnes, voir figure 2.6.

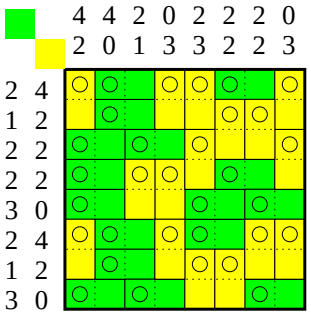


Figure 2.6. Un pavage par des dominos et ses projections.

Reconstruction de Pavage (Complet) de Dominos

Entrée. des entiers r_{ai}, c_{aj} pour tout $a \in \{1,2\}$ et $i,j \in \{1,...,n\}$

Sortie. un pavage de dominos P tel que pour tout $i,j \in \{1,...,n\}$, la ligne i contienne r_{1i} marques de dominos horizontaux et r_{2i} marques de dominos verticaux et la colonne j contienne c_{1j} marques de dominos horizontaux et c_{2i} marques de dominos verticaux.

Beaucoup de personnes ont étudié ce problème qui reste ouvert malgré tout. Par contre il existe un algorithme qui résout un cas particulier: Il s'agit du cas où aucun domino n'est à cheval entre des colonnes $2i+1$ et $2i+2$ pour i entier. Ce cas est en fait équivalent au problème de reconstruction de pavage (complet) par des dominos verticaux et des cellules (carrés de taille 1×1). Dans ce cas un algorithme de reconstruction glouton très simple — similaire à celui pour le problème des matrices 0-1 — résout le problème. Ceci a été observé indépendamment par nous [38] et par Christophe Picouleau [66].

Eric Golès et Ivan Rapaport [38] ont étudié le problème plus général où les tuiles sont deux rectangles (barres) de dimension $1 \times h$ et $w \times 1$ respectivement. Ils ont trouvé un algorithme polynomial dans le cas où les projections sont identiques pour toutes les lignes. Ma contribution dans [38] consistait à montrer qu'un problème plus général est NP-complet. Il s'agit toujours de paver une grille avec des dominos, en respectant des nombres donnés de dominos verticaux et horizontaux dans chaque ligne et colonne. Mais cette fois ci, l'entrée du problème spécifie aussi les zones interdits de la grille qui ne doivent pas être couvertes de dominos. La preuve consiste en une réduction du problème de placement de jetons à 3 couleurs. Dans ce dernier problème chaque case ne peut avoir que quatre états : vide, ou alors contenant un jeton des couleurs 1, 2 ou 3. Le gadget de la réduction est une grille carrée d'une taille 7×7 avec des zones interdites. Il n'y a que quatre manières de paver la grille avec des dominos, que nous faisons correspondre aux quatre états d'une cellule du premier problème. Les projections des quatre pavages sont des vecteurs deux à deux orthogonaux, qui assure qu'il y ait une bijection entre les ensembles de solutions des deux problèmes.

2.2.4 Classification d'ensembles de tuiles

Pour une vision unifiée, considérons le problème de placement de jetons colorés comme un problème de reconstruction de pavage, avec différents types de pavés de taille 1×1. Se pose alors une question très naturelle. Pour quels types de pavés le problème de reconstruction est-il polynomial et pour quel type de pavés le problème est-il NP-complet ? Il sera sans doute difficile de répondre à cette question dans sa généralité, car le problème est ouvert pour certains pavés élémentaires, comme les dominos. Pour l'anecdote, le premier type de pavé pour lequel j'ai pu montrer la NP-complétude du problème avait la forme de la croix gammée, ce qui m'a alors immédiatement forcé de trouver des formes de remplacement. J'ai

toutefois essayé d'apporter une réponse en étudiant le problème pour différents types de pavés. Je me suis alors rendu compte que la réduction décrite ci-haut pouvait s'appliquer à d'autres ensembles de tuiles. Pour différents ensembles de tuiles il fallait alors exhiber quatre pavages d'une grille de taille constante, dont les projections seraient linéairement indépendantes, et prouver que lors de notre réduction seulement ces pavages pouvaient apparaître. Les résultats de cet article [32] sont résumés dans le tableau 2.2.

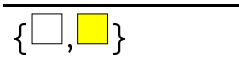
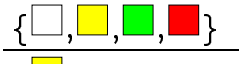
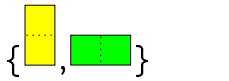
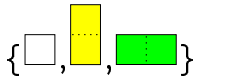
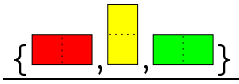
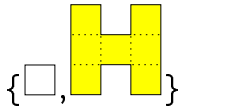
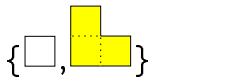
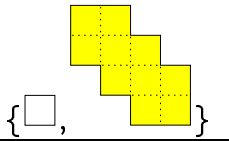
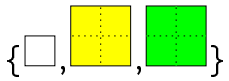
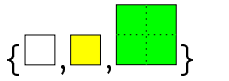
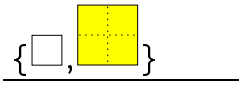
Type de pavés	Complexité	Référence
	$O(n)$	[67]
	ouvert	¨Pb. des jetons à 2 couleurs¨
	NP-complet	[34]
	ouvert	
	$O(n^2)$	[38] et [66]
	$\succeq$ Pb. des jetons à 2 couleurs	[32]
	NP-complet	[32]
	NP-complet	[44]
	ouvert	
	NP-complet	[32]
	NP-complet	[32]
	NP-complet	[32]
	NP-complet	[32]
	ouvert	

Tableau 2.2. Complexité du problème de reconstruction de pavage pour différents types de pavés. Nos contributions sont marquées en gris.

A titre d'exemple nous détaillons une de ces preuves.

Théorème 2.3. *Le problème de pavage sous contraintes tomographiques pour l'ensemble de tuiles D:={□,▭,▯} est NP-complet.*

Démonstration. La preuve est une réduction du problème de jetons de 3 couleurs. Soit donc $(r_{i,k}, c_{j,k})$ une instance de ce dernier problème pour $i, j \in [n]$ et $k \in \{1, 2, 3\}$, où 1, 2, 3 représentent les 3 couleurs, et pour plus de confort nous ajoutons les projections des cases vides. Donc $r_{i,0} = n - r_{i,1} - r_{i,1} - r_{i,2}$ pour tout $i \in [n]$ et $c_{j,0}$ est défini de la même manière. Le gadget de cette preuve est une grille 3×3 pavé de 4 manières différentes, voir figure 2.7.

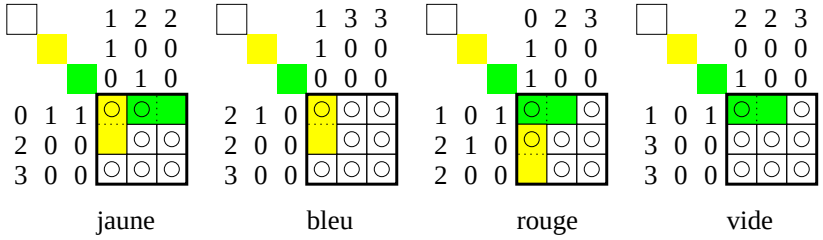


Figure 2.7. Le gadget de la réduction.

Les projections sur les lignes de ces pavages sont des matrices linéairement indépendantes :

$$\boldsymbol{r}_{\text{jaune}} = \begin{pmatrix} 0 & 1 & 1 \\ 2 & 0 & 0 \\ 3 & 0 & 0 \end{pmatrix}, \boldsymbol{r}_{\text{bleu}} = \begin{pmatrix} 2 & 1 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 0 \end{pmatrix}, \boldsymbol{r}_{\text{rouge}} = \begin{pmatrix} 1 & 0 & 1 \\ 2 & 1 & 0 \\ 2 & 0 & 0 \end{pmatrix}, \boldsymbol{r}_{\text{vide}} = \begin{pmatrix} 1 & 0 & 1 \\ 3 & 0 & 0 \\ 3 & 0 & 0 \end{pmatrix}.$$

Il en est de même pour les projections sur les colonnes $\boldsymbol{c}_{\text{jaune}}, \boldsymbol{c}_{\text{bleu}}, \boldsymbol{c}_{\text{rouge}}, \boldsymbol{c}_{\text{vide}}$.

Nous définissons maintenant une instance pour le problème de pavage pour l'ensemble de tuiles D. Soit r', c' des vecteurs de $[3n] \times \{0, 1, 2\}$, où 0,1,2 représentent respectivement les tuiles □,▭,▯. Pour tout $i \in [n]$ nous définissons

$$\begin{pmatrix} r'_{3i,0} & r'_{3i,1} & r'_{3i,2} \\ r'_{3i+1,0} & r'_{3i+1,1} & r'_{3i+1,2} \\ r'_{3i+2,0} & r'_{3i+2,1} & r'_{3i+2,2} \end{pmatrix} := r_{i,0}\boldsymbol{r}_{\text{jaune}} + r_{i,1}\boldsymbol{r}_{\text{bleu}} + r_{i,2}\boldsymbol{r}_{\text{rouge}} + r_{i,3}\boldsymbol{r}_{\text{vide}}.$$

Les projections c' sont définies de la même manière.

Maintenant il est clair par la construction que si l'instance (r, c) admet une solution S alors l'instance (r', c') admet aussi une solution, tout simplement en remplaçant dans S chaque case par une des quatre grilles 3×3 correspondante. Pour prouver l'autre direction, soit S' une solution de (r', c'). Nous allons montrer que (r, c) admet aussi une solution.

Le pavage S' est décomposé en n^2 blocs de taille 3×3. La première partie de la preuve consiste à montrer que chaque bloc est pavé dans une des quatre configurations de la figure 2.7. C'est cette partie de la preuve qui est différente d'un ensemble de tuiles à un autre. Ensuite la deuxième partie en est indépendante.

Pour notre ensemble de tuiles, cette première partie est assez simple. Observons d'abord que les projections (r', c') forcent à ce que dans dans S' aucun domino ne soit à cheval entre deux blocs. Ensuite soit $I = \{i \in [3n] : i \bmod 3 = 0\}$. Alors de nouveau les projections (r', c') forcent $I \times I$ à être complètement libre de la tuile blanche $\square$, et $\overline{I} \times \overline{I}$ à être entièrement pavé de la tuile blanche. Ceci peut être montré en utilisant la proposition 2.1. Les seuls pavages de blocs compatibles avec ces observations sont les 4 décrits de la figure 2.7. Soit S le pavage d'une grille $n \times n$ obtenu en remplaçant chaque bloc, par une cellule, éventuellement muni d'un jeton de la couleur correspondante.

La deuxième partie de la preuve consiste à prouver que S a les projections (r, c) requises. Considérons la i-ème ligne. Soient a_1, a_2, a_3 le nombre de jetons de couleurs respectivement 1,2,3 dans cette ligne, et $a_0 = n - a_1 - a_2 - a_3$ le nombre de cellules vides. Alors par construction nous avons

$$a_0 \boldsymbol{r}_{\text{jaune}} + a_1 \boldsymbol{r}_{\text{bleu}} + a_2 \boldsymbol{r}_{\text{rouge}} + a_3 \boldsymbol{r}_{\text{vide}} = r_{i,0} \boldsymbol{r}_{\text{jaune}} + r_{i,1} \boldsymbol{r}_{\text{bleu}} + r_{i,2} \boldsymbol{r}_{\text{rouge}} + r_{i,3} \boldsymbol{r}_{\text{vide}}.$$

Par indépendance linéaire des matrices $\boldsymbol{r}_{\text{jaune}}$, $\boldsymbol{r}_{\text{bleu}}$, $\boldsymbol{r}_{\text{rouge}}$, $\boldsymbol{r}_{\text{vide}}$ nous avons que $a_k = r_{i,k}$ pour tout $k \in \{0, 1, 2, 3\}$. La projection sur les lignes de S est donc r. Le cas des colonnes est identique. Ceci prouve que S est bien une solution à (r, c) $\square$

2.3 Problèmes ouverts

Les problèmes ouverts sont nombreux. Par exemple il serait souhaitable de connaître une caractérisation des tuiles t pour lesquelles le problème de reconstruction de $\{\square, t\}$ est polynomial. Nous conjecturons que le problème est NP-complet quand t est une tuile qui s'*emboîte*. Formellement nous disons qu'une tuile t de largeur w et hauteur h s'emboîte si la grille $(2w - 1) \times (2h - 1)$ est pavable avec deux tuiles t. Nous avons pu le prouver pour une série de telles tuiles sans néanmoins pouvoir en dégager une preuve générale. En particulier le problème reste ouvert pour la tuile en forme de la lettre H, voir tableau 2.2. Pour les tuiles qui ne s'emboîtent pas le problème est déjà ouvert pour la tuile carré 2×2.

Ce problème consiste à placer des jetons sur la grille en respectant des contraintes du nombre de jetons dans chaque ligne et colonne avec la contrainte supplémentaire que deux jetons ne doivent pas être adjacent. Ceci nous amène à un autre problème issu du contrôle aérien.

> Soit la grille $n \times n$ munie de la distance Δ induit par la norme $L_{\max}$.
> Étant donné m points $p_1, \ldots, p_m$ sur la grille avec la propriété (*) $\forall i, j\ i \neq j$: $\Delta(p_i, p_j) > 1$, est-ce qu'il existe m points $q_1, \ldots, q_m$ avec $\forall i$: $\Delta(p_i, q_i) = 1$ et de nouveau (*) $\forall i, j\ i \neq j$: $\Delta(q_i, q_j) > 1$?

L'idée est qu'on dispose de m avions sur un plan, respectant tous une distance de sécurité les uns des autres, et on veut pouvoir les déplacer chacun d'une case de la grille, tout en respectant les distances de sécurité.

Chapitre 3
Algorithmes quantiques

Ce chapitre est basé sur les articles suivants :

1. [30] *Quantum algorithms for element distinctness*, avec H. Buhrman, M. Heiligman, P. Høyer, F. Magniez, M. Santha et R. de Wolf. Dans IEEE Conference on Computational Complexity, pp. 131–137, 2001. A paraître dans SIAM Journal on Computing.
2. [39] *Quantum query complexity of some graph problems*, avec M. Heiligman, P. Høyer et M. Mhalla. Dans Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP), pp. 481–493, 2004.
3. [9] *Quantum query complexity in computational geometry revisited*, avec A. Bahadur, R. Kulkarni et T. Lafaye, manuscrit, 2005.

3.1 Préliminaires

L'idée de construire un ordinateur qui tirerait profit des effets de la mécanique quantique a été proposée dans les années 80 par Richard Feynman [42, 43]. Il fallut attendre la fin des années 90 pour que le modèle de calcul quantique soit formellement défini. Dans ce contexte, ma thèse de doctorat définissait et étudiait un modèle d'automates cellulaires quantiques [37]. Puis le domaine a véritablement pris son envol par le résultat de Peter Shor, qui montra qu'un ordinateur quantique peut factoriser un entier donné en temps polynomial espéré [71]. Ce résultat est très important car d'une part il rend vulnérable le système de cryptographie RSA dont la sécurité est basée sur l'hypothèse qu'il n'existe pas d'algorithme polynomial pour le problème de factorisation. Il est aussi important d'autre part, parce qu'il donne un algorithme quantique à erreur bornée qui est exponentiellement plus rapide que le meilleur algorithme classique connu pour un problème réel, alors qu'avant une telle séparation n'avait été montrée que pour des problèmes artificiels avec promesse sur les entrées. Contrairement à ce que beaucoup de personnes en dehors du domaine croient, l'ordinateur quantique n'apporte une accélération exponentielle que pour certains problèmes, pour d'autres il n'apporte qu'une accélération polynomiale et pour d'autres aucune. Un de nos buts est justement de déterminer la complexité quantique des problèmes.

Un nombre incroyable d'introductions au calcul quantique ont été écrites, aussi je ne vais pas en donner une de plus ici, mais référencer vers [65, 36]. Nous allons juste introduire les notions minimales pour comprendre ce chapitre. L'état d'un ordinateur quantique, dans ce modèle simplifié, est un vecteur $\varphi \in \mathbb{C}^N$ de norme 1 pour la norme L_2. Ce vecteur est appelé une *superposition*. L'espace des vecteurs est appelé *l'espace de Hilbert*, et les vecteurs de base sont notés $|i\rangle$ pour $i \in [N]$. La superposition φ est alors aussi notée $|\varphi\rangle = \sum_{i\in[N]} \alpha_i |i\rangle$ où α_i sont des nombres complexes appelés *amplitudes*. Il y a essentiellement deux choses qu'on peut faire avec une superposition. Premièrement on peut y appliquer un opérateur linéaire unitaire $U \in \mathbb{C}^{N\times N}$, qui transforme $|\varphi\rangle$ en une nouvelle superposition $U|\varphi\rangle$. Pas toutes les matrices unitaires sont raisonnables, car elles doivent pouvoir être implémentables par un nombre polynomial d'opérations élémentaires locales. Le fait que l'opérateur doive être unitaire n'est pas vraiment une restriction, car toute fonction f peut être calculée de manière réversible (unitaire). Une deuxième chose qu'on peut faire avec une superposition est de l'observer. On observe alors i avec probabilité $|\alpha_i|^2$, suite à quoi la superposition devient $|i\rangle$.

3.1.1 Complexité de requêtes

La contribution des informaticiens consiste à établir la différence entre le pouvoir de calcul d'un ordinateur quantique et d'un ordinateur classique pour différents problèmes et différents modèles. Je me suis intéressé à la complexité en requêtes de certains problèmes importants pour l'informatique. La complexité en requêtes est une autre mesure que la complexité en temps, mais il est possible de prouver des bornes inférieures sur la complexité en requêtes, alors que ceci est pratiquement impossible pour la complexité en temps. Ainsi peut-on éclairer une séparation entre les complexités quantique et classique d'un problème. La complexité de requêtes est aussi parfois appelée *profondeur d'un arbre de décision*. Dans ce modèle l'entrée $x \in \{0,1\}^n$ est donnée sous forme d'un oracle, qu'il faut questionner pour pouvoir calculer une fonction $f(x)$. Pour questionner un bit de l'entrée, on doit faire appel à un opérateur de requête F: $|b, i, w\rangle \mapsto (-1)^{bx_i}|i, w\rangle$. Ici $|b, i, w\rangle$ est un vecteur de base de l'espace de Hilbert de la machine, qui comporte trois registres. Le premier comporte un bit de contrôle, le deuxième l'indice à questionner, et le troisième, le registre de travail. Un algorithme quantique à T requêtes consiste alors à appliquer (dans cet ordre) sur l'état initial $|1, 0, 0\rangle$ des opérateurs $O_0, F, O_1, F, ..., F, O_T$. Puis on observe les derniers qubits du registre de travail, et on affiche le résultat. Un algorithme quantique à erreur bornée doit alors fournir la bonne réponse $f(x)$ avec une probabilité au moins $2/3$. Notez que cette probabilité peut être rendue arbitrairement petite, disons ϵ, avec $O(\log(1/\epsilon))$ répétitions et $O(\sqrt{\log(1/\epsilon)})$ répétition dans certains cas, détaillés plus loin.

Pour comparer les complexités de requêtes quantiques et classiques, on utilise les notations suivantes.

- $D(f)$ est le nombre minimal de requêtes d'un algorithme classique, déterministe qui calcule f.
- $R_2(f)$ est le nombre minimal de requêtes d'un algorithme classique, probabiliste à erreur bornée, c'est-à-dire qui sur l'entrée x affiche $f(x)$ avec probabilité au moins $2/3$.
- $Q_E(f)$ est le nombre minimal de requêtes d'un algorithme quantique, qui sur l'entrée x du domaine de f affiche $f(x)$ avec certitude.
- $Q_2(f)$ est le nombre minimal de requêtes d'un algorithme quantique à erreur bornée.

Pour des problèmes à promesses, ou plus généralement des fonctions partielles f, le fossé quantique/classique peut être exponentiel. Il existe par exemple une fonction $f\colon \{0,1\}^{2^n} \to \{0,1\}$ — le problème de Simon — pour laquelle $Q_E(f) = n$ [23] et $R_2(f) \in \Omega(2^n)$ [54].

Considérons plus des fonctions totales f. Il a été montré que le fossé est au plus un polynôme de degrée 6, $R_2(f) \in O(Q_2(f)^6)$, et on conjecture que ce fossé est au plus quadratique [19]. Pour certaines fonctions il n'y a pas de fossé du tout, en fait pour tout $0 \leq \alpha \leq 1$ il existe une fonction f, telle que $Q_2(f) \in \Theta((R_2(f))^{1/(1+\alpha)})$. Pour des fonctions monotones, la méthode de borne inférieure sur $Q_2(f)$ est plus forte et montre que le fossé est au plus un polynôme de degré 4 [19]. Pour les calculs sans erreur il a été montré que le fossé est au plus un polynôme de degré 3 [62].

3.1.2 Algorithme de recherche de Grover

L'ingrédient quantique des algorithmes de ce chapitre est essentiellement l'algorithme de recherche quantique de Lov Grover [49, 22]. Le problème est le suivant, étant donné $f\colon [N] \to \{0,1\}$ on veut trouver un indice $x \in [N]$ tel que $f(x) = 1$, en utilisant un nombre minimal de requêtes à f. Classiquement, $\Omega(N)$ requêtes sont nécessaires pour un algorithme classique à erreur bornée.

L'algorithme de Grover trouve une solution avec probabilité au moins $1/2$, sous l'hypothèse que f n'est pas constant 0 avec $O(\sqrt{N})$ requêtes à f. Une autre variante trouve une solution avec un nombre espéré de requêtes $O(\sqrt{N/t})$ où t est le nombre de solutions ($t = |f^{-1}(1)|$), sous l'hypothèse que $t > 0$ et ne termine jamais si f est constant 0. Le nombre t ne doit pas forcément être connu par l'algorithme.

Une dernière variante est *l'amplification d'amplitude* [24]. Ici pour un problème donné on dispose d'un algorithme A qui avec probabilité au plus $1/2$ affiche « *je ne sais pas* » et affiche la bonne réponse sinon. Il faut que l'algorithme soit implémenté par une matrice unitaire U_A, ce qui veut dire qu'en particulier A ne doit pas effectuer de mesure. Classiquement pour diminuer la probabilité d'erreur à ε il faudrait effectuer $\log(1/\varepsilon)$ répétitions de A, et l'amplification d'amplitudes le réalise avec seulement $O(\sqrt{\log(1/\varepsilon)})$ appels à U_A et $U_A^\dagger$.

Notez que cette technique ne permet pas de réduire la probabilité d'erreur de n'importe quel algorithme. Par exemple considérons le problème où on dispose d'une fonction f et on veut trouver un indice i tel que $f(i)$ soit minimum. Soit un algorithme A qui afficherait un tel indice j qu'avec probabilité au moins $1/2$, $f(j)$ est minimum. Par contre une fois j affiché, il n'y pas de moyen simple de vérifier le résultat, c'est-à-dire de produire une sortie qui soit soit exacte soit « *je ne sais pas* ».

3.1.3 Les bornes inférieures

Pour prouver des bornes inférieures sur la complexité de requêtes en erreur bornée d'un problème, plusieurs méthodes s'offrent à nous: la méthode d'adversaire quantique d'Andris Ambainis [3], la méthode par polynômes [19], la méthode de complexité de Kolmogorov [58], et la méthode par programmation semi-définie [17]. Les méthodes les plus générales sont les plus fortes, mais aussi les plus difficiles à appliquer, dans de nombreux cas on n'arrive pas à prouver les propositions nécessaires. Au final, les méthodes les plus utilisées sont la méthode par adversaire quantique et la méthode par polynômes. Elles sont incomparables, dans le sens où y a des problèmes pour lesquels il est impossible de prouver des bornes optimales avec la première méthode et des problèmes où la deuxième méthode ne peut pas fournir de bornes inférieures optimales [4].

Toutes les bornes inférieures que j'ai prouvées utilisent la première méthode, qui est résumée ici. Pour plus de simplicité nous la décrivons pour des fonction booléennes, mais elle s'applique à des fonction générales. Donc soit L un problème de décision. Soient des sous-ensembles $A \subseteq L$ et $B \subseteq \overline{L}$ et une relation $R \subseteq A \times B$, qui relie des instances positives et négatives de même taille, $xRy \Rightarrow |x| = |y|$. Soient $m, m', l_{x,i}, l'_{y,i}$ des fonctions entières satisfaisant

$$\forall x \in A: |\{y \in B: xRy\}| \geqslant m(|x|)$$
$$\forall y \in B: |\{x \in A: xRy\} \geqslant m'(|y|)$$
$$\forall x \in A \forall i \in [1,n]: |\{y \in B: xRy, x_i \neq y_i\}| \leqslant l_{x,i}$$
$$\forall y \in B, \forall i \in [1,n]: |\{x \in A: xRy, x_i \neq y_i\}| \leqslant l'_{y,i}.$$

On définit $l_{\max}(n) = \max l_{x,i} l'_{y,i}$, où le maximum est pris sur $x, y, i, |x| = |y| = n$ avec xRy et $x_i \neq y_i$.

Théorème 3.1. *[3] La complexité quantique en erreur bornée de L est* $\Omega(\sqrt{mm'/l_{\max}})$.

Pour donner des exemples d'applications, soit la fonction OU qui calcule le ou logique des n bits en entrée. On pose $A = \{0...0\}$, $B = \{10...0, 01...0, ..., 00...1\}$ et $R = A \times B$. Alors $m = n$, $m' = 1$ et $l'_{y,i} = l_{x,i} = 1$, d'où $Q_2(\text{OU}) = \Omega(\sqrt{N})$. Pour la PARITÉ, on pose $A = \{x \in \{0,1\}^n : \bigoplus x_i = 1\}$, $B = \{x \in \{0,1\}^n : \bigoplus x_i = 0\}$ et xRy pour $x \in A$, $y \in B$ si et seulement si x et y diffèrent en exactement un bit. Alors $m = m' = n$ et de nouveau $l'_{y,i} = l_{x,i} = 1$, d'où $Q_2(\text{PARITÉ}) = \Omega(N)$.

3.2 Contributions

3.2.1 Le problème d'éléments distincts

Il est intéressant d'établir la complexité en requêtes précises pour différentes fonctions. C'est ce que nous avons entrepris pour des fonctions importantes. Voici le PROBLÈME D'ELÉMENTS DISTINCTS: on dispose d'une fonction $f : [N] \to [M]$ avec $M \geq N$ et il s'agit de décider si f est injectif ou s'il existe deux indices distincts $i, j \in [N]$ tel que $f(i) = f(j)$, et dans ce cas on veut fournir une telle paire de collision. On peut imaginer f comme une fonction de signature et on veut qu'il soit difficile de trouver deux documents qui ont la même signature, ce qui serait une atteinte à la sécurité du système de signature.

Il existe un problème lié qui porte le nom de PROBLÈME DE COLLISION, motivé par le fait qu'en pratique on préfère des signatures qui sont beaucoup plus petites que le document lui-même. Dans ce cas évidement la fonction n'est pas injective. De nouveau on aimerait qu'il soit difficile de trouver une paire de collision, bien que l'existence soit établie. Formellement dans le problème de collision on dispose d'une fonction f *two-to-one*, c'est-à-dire qui est telle que pour tout $i \in [N]$ il existe exactement un autre indice $j \in [N] \setminus \{i\}$ tel que $f(i) = f(j)$. Et le but est de fournir une telle paire de collision.

Le paradoxe des anniversaires fournit une relation très simple entre les deux problèmes. Un sous-ensemble uniformément aléatoire de taille $\sqrt{N}$ du domaine $[N]$ d'une fonction two-to-one contient avec grande probabilité une paire de collision. Donc un algorithme (à erreur bornée) pour le problème d'éléments distincts à $O(N^\alpha)$ requêtes implique un algorithme pour le problème de collision à $O(N^{a/2})$ requêtes. Et en effet la complexité des deux problème est quadratiquement liée: La complexité classique du problème d'éléments distincts est $\Theta(N)$ alors celle du problème de collision est $\Theta(N^{1/2})$. Et récemment il a été établit que la complexité quantique du premier problème est $\Theta(N^{2/3})$ alors que celle du deuxième est $\Theta(N^{1/3})$.

En 1998 Gilles Brassard, Peter Høyer et Alain Tapp ont donné un algorithme quantique à erreur bornée qui résout le problème de collision avec $O(N^{1/3})$ requêtes [25]. En 2002 Yaoyun Shi a donné une borne inférieure de $\Omega(n^{1/3})$ montrant que l'algorithme de Brassard et al. bien qu'étant très simple est optimal [2].

Avec Harry Buhrman, Mark Heiligman, Peter Høyer, Frédéric Magniez, Miklos Santha et Ronald de Wolf [30] nous avons étudié entre autre le problème d'éléments distincts, et fourni un algorithme quantique à erreur bornée qui utilise $O(N^{3/4}\log n)$ requêtes dans le pire des cas, alors qu'une réduction triviale du problème OU établit seulement une borne inférieure de $\Omega(N^{1/2})$. Ce nombre important de co-auteurs s'explique par la fusion de plusieurs articles qui furent écrits indépendamment. Depuis cet article, beaucoup de recherches ont été entreprises pour déterminer la complexité exacte du problème. Par la remarque ci-haut, la borne inférieure de Yaoyun Shi pour le problème de collision implique une borne inférieure de $O(N^{2/3})$ pour le problème d'éléments distincts. Puis un algorithme optimal a été trouvé par Andris Ambainis en 2003, sur la base d'une marche aléatoire quantique [5].

L'article avec H. Buhrman et al. donnait aussi des algorithmes pour trouver l'intersection de deux tableaux triés, ou pour trouver un triangle dans un graphe. Notre algorithme était une application très simple de l'algorithme de recherche quantique.

Algorithme pour trouver une paire de collision de $f\colon [N] \to [M]$

1. Partitionner le domaine de f en $\sqrt{N}$ ensembles disjoints $S_1, \ldots, S_{\sqrt{N}}$ de taille $O(\sqrt{N})$ chacun. On dit que S_k est bon s'il contient un indice i qui est en collision avec un autre indice $j \in [N]$.

2. Appliquer l'amplification d'amplitudes sur la *procédure interne* suivante.

   a) Choisir un ensemble S_k au hasard uniformément.

   b) Questionner toutes les valeurs $f(i)$ pour $i \in S_k$ et construire un arbre binaire de recherche sur l'ensemble $f(S_k) := \{f(i)\colon i \in S_k\}$. Si S_k contient une paire de collision, l'afficher.

   c) Sinon chercher un élément $j \in [N] \backslash S_k$ tel que $f(j) \in f(S_k)$. Pour cela utiliser la recherche quantique qui réussit avec probabilité au moins $1/2$ dans le cas où S_k est bon. Dans le cas positif, afficher la paire de collision.

Théorème 3.2. *L'algorithme ci-haut trouve une paire de collisions en $O(N^{3/4})$ requêtes espérées, si f n'est pas injectif.*

Démonstration. L'étape 2c est implémentée par la variante de l'algorithme de recherche qui trouve une solution avec probabilité au moins $1/2$ en $O(\sqrt{N})$ requêtes, dans le cas où S_k est bon. La probabilité qu'en 2a un tel ensemble est choisi est au moins $1/\sqrt{N}$. Donc les pas 2a-c réussissent avec probabilité $1/2\sqrt{N}$ au moins et l'amplification d'amplitudes doit effectuer $O(N^{1/4})$ répétitions de cette procédure, ce qui complète l'analyse. □

Nous avions en vain essayé de prouver une borne inférieure pour ce problème et récemment il a été montré que la méthode d'adversaire d'Ambainis est limitée pour ce problème et ne permet pas de prouver la borne optimale [58]. Nous avons donc seulement réussi à prouver qu'il faut $\Omega(N)$ requêtes pour des problèmes plus généraux, comme décider de la parité du nombre de collisions.

3.2.2 Recherche de minima

En 1996, avec Peter Høyer nous avions trouvé que l'algorithme de Grover, en plus de trouver un bon élément d'un tableau, permettait de trouver celui qui minimise une fonction de coût donnée [37]. Au bout de $O(\sqrt{N})$ requêtes l'algorithme donne la bonne solution avec grande probabilité. Il était décrit dans ma thèse de doctorat, mais n'avait pas été publié à l'époque. Ce n'est que récemment que nous avons trouvé une généralisation utile pour résoudre certains problèmes de graphes.

L'algorithme de recherche du minima, tout simplement, fait appel à la recherche de Grover pour trouver un bon élément qui améliore le coût de l'élément courant. Le résultat vient remplacer l'élément courant et on recommence. Il existe un nombre espéré de requêtes pour que l'élément courant soit la solution. On arrête alors cette procédure au bout de deux fois ce nombre, ce qui donne une grande probabilité de succès.

Notre généralisation était motivée par le problème suivant. Supposons qu'on dispose de k fonctions $f_1, ..., f_k$ chacune d'un domaine de taille n/k, et on veut trouver des indices $i_1, ..., i_k$ tels qu'avec grande probabilité pour tout $1 \leqslant a \leqslant k$, $f_a(i_a)$ soit minimum pour f_a. Si on fait appel bêtement k fois à l'algorithme de recherche du minima, il faut que chaque résultat soit la bonne solution avec probabilité au moins $1 - 1/k$. Il faudrait alors $O(\log k)$ répétitions pour atteindre cela. Ceci peut être contourné tout simplement en effectuant une recherche conjointe sur les k fonctions. Dans ce cas nous recherchons dans un espace commun de taille n un indice qui améliore un champ de l'élément courant qui est maintenant un k-uplet d'indices. Pour cela nous considérons les généralisations suivantes.

Trouver les d plus petites valeurs d'une fonction. Étant donné une fonction $f\colon [N] \to \mathbb{N}^\star$, où $\mathbb{N}^\star := \mathbb{N} \cup \{\infty\}$ et un entier d on veut trouver les indices des d plus petits éléments, donc un ensemble $I \subseteq [N]$ de cardinalité d tel que pour tout $j \in [N] \backslash I$ nous ayons $f(i) \leqslant f(j)$ pour tout $i \in I$.

Trouver d éléments de type différent. Étant donné une fonction $g: [N] \to \mathbb{N}$ qui donne à chaque indice un type parmi e possibles, $e := |\{g(i) : i \in [N]\}|$, et un entier d', on veut trouver $d = \min\{e, d'\}$ et un sous ensemble $I \subseteq [N]$ de cardinalité d tel que $g(i) \neq g(j)$ pour tout $i, j \in I$, $i \neq j$.

Trouver d plus petits éléments de type différent. Étant données deux fonctions f, g et un entier d' on veut trouver $d = \min\{e, d'\}$ et un sous ensemble $I \subseteq [N]$ de cardinalité d tel que d'une part $g(i) \neq g(j)$ pour tout $i, j \in I, i \neq j$ et tel que pour tout $j \in [N] \backslash I$ et tout $i \in I$ si $f(j) < f(i)$ alors il existe un autre indice $i' \in I$ avec $f(i') \leqslant f(j)$ et $g(i') = g(j)$.

Ces problèmes sont illustrés en figure 3.1.

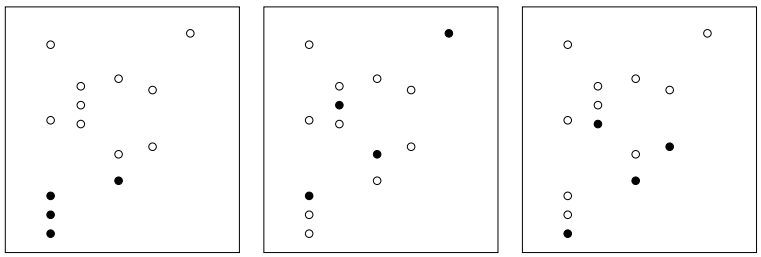


Figure 3.1. Illustration des trois problèmes. Chaque indice $i \in [N]$ est représenté par un point avec coordonnée $g(i)$ et abscisse $f(i)$. (a) 4 points de valeurs minimum, (b) 4 points de type différent, (c) 4 points de valeur minimum et de type différent.

Nous avons obtenu le résultat suivant. Nous avons découvert plus tard qu'un algorithme pour le problème 1 avait déjà été trouvé par Nayak et Wu [64] puis rendu optimal par Nayak [63]. Sa méthode est différente de la nôtre.

Théorème 3.3. *La complexité quantique à erreur bornée des trois problèmes ci-dessus est* $\Theta(\sqrt{dN})$.

La borne supérieure est similaire à l'algorithme initial pour rechercher le minimum d'une fonction donnée. Pour la borne inférieure nous avons utilisé la méthode d'adversaire d'Andris Ambainis. Maintenant nous allons voir des applications de ces algorithmes.

Pour des problèmes de graphes, on considère deux modèles de requêtes. Dans le modèle de matrices d'adjacence, le graphe est donné sous forme d'une matrice booléenne M $n \times n$, qui est symétrique dans le cas non-orienté. Dans le cas pondéré M est une matrice sur $\mathbb{N}^\star$ où $M(u, v)$ est le poids de l'arête (u, v) si elle existe et ∞ sinon.

Le modèle de tableau d'adjacence est similaire à la représentation courante par liste d'adjacence avec la différence qu'en une seule requête on peut accéder en temps constant au i-ème voisin d'un sommet u. Dans ce modèle, on dispose des degrés d_u de chaque sommet u, et des fonctions $f_u\colon [d_u] \to V$ qui contiennent dans un ordre arbitraire les voisins d'un sommet u. Dans le cas pondéré, la fonction renvoie en plus du voisin le poids de l'arête, $f_u\colon [d_u] \to V \times \mathbb{N}$.

Nous avons obtenu les bornes suivantes sur quatre problèmes de graphes dans les deux modèles.

Problème	Modèle de matrice		Modèle de tableau
Arbre de recouvrement minimal	$\Theta(n^{3/2})$		$\Theta(\sqrt{nm})$
Connexité	$\Theta(n^{3/2})$		$\Theta(n)$
Connexité forte	$\Theta(n^{3/2})$		$\Omega(\sqrt{nm})$, $O(\sqrt{nm}\log n)$
Plus courts chemins	$\Omega(n^{3/2})$,	$O(n^{3/2}\log^2 n)$	$\Omega(\sqrt{nm})$, $O(\sqrt{nm}\log^2 n)$
Tester 2-colorabilité	$\Omega(n^{3/2})$[20]	$O(n^{3/2})$	$\Theta(n)$[39]
Tester existence triangle	$\Omega(n)$,	$O(n^{1.3})$ [61]	
Couplage maximal	$\Omega(n^{3/2})$		

Tableau 3.1. Complexités quantiques à erreur bornée de différents problèmes de graphes. En comparaison, les complexités classiques probabilistes pour ces problèmes sont $\Theta(n^2)$ dans le modèle de matrice et $\Theta(n+m)$ pour le modèle de tableau d'adjacence. Nos contributions sont marquées en gris.

Il y a plusieurs choses à observer sur ces résultats. D'abord pour les graphes denses, où $m = \Theta(n^2)$, les complexités dans les deux modèles sont à un facteur polylog près les mêmes, à l'exception du problème de connexité. Ceci est étonnant, car les modèles sont toutefois différents dans le cas dense : le test $(u, v) \in E$, qui coûte une seule requête dans le modèle de matrice, vaut $\Theta(\sqrt{\min\{d_u, d_v\}})$ requêtes dans le modèle de tableau, ce qui est moralement $\Theta(\sqrt{n})$ pour les graphes denses.

La complexité du problème de connexité dans le modèle de tableau ne dépend pas du nombre d'arêtes. L'intuition est que dans ce cas, la taille de l'espace de recherche augmente avec le nombre d'arêtes, mais le nombre d'arbres de recouvrement aussi, et ceci s'équilibre. La complexité en temps de ces algorithmes est à un facteur logarithmique la même que la complexité en requête.

3.2.3 Arbre de recouvrement minimal

Dans le problème d'arbre de recouvrement minimal, on dispose d'un graphe avec des arêtes pondérées et on cherche un ensemble maximal d'arêtes qui soit sans cycle (une forêt recouvrante en fait) mais qui minimise le poids total. Pour simplifier les notations, on suppose que le graphe est connecté et on cherche alors un arbre couvrant de plus petit poids. L'algorithme de Boruvka [21] pour ce problème repose sur l'observation suivante.

Remarque 3.4. Soit $U \subseteq V$ un sous-ensemble de sommets d'un graphe connecté $G(V, E)$ et $e \in (U \times \overline{U}) \cap E$ une arête sur le bord de U de poids minimal. Alors il existe un arbre de recouvrement minimal contenant e.

La preuve consiste tout simplement à ajouter e dans un arbre de recouvrement minimal dans le cycle ainsi créé, d'enlever l'autre arête e' qui fait partie de $(U \times \overline{U}) \cap E$. Clairement le résultat est aussi un arbre de recouvrement minimal.

L'algorithme quantique que nous proposons utilise exactement cette remarque. On dispose d'un certain nombre d'arbres $T_1, ..., T_\ell$ qui couvrent le graphe et tel que pour tout $1 \leqslant a \leqslant \ell$ l'arbre T_a est minimal pour tous les arbres couvrants $V(T_a)$. Initialement $\ell = n$ et chaque arbre consiste en un unique sommet isolé. A chaque itération on cherche alors un ensemble A d'arêtes de bord pour chaque ensemble $V(T_a)$. Ces arêtes sont alors ajoutées aux arbres, ce qui a pour effet de fusionner chaque arbre avec au moins un autre. La procédure est répétée jusqu'à ce qu'il ne reste qu'un seul arbre couvrant minimal du graphe. Pour l'itération, on utilise exactement la procédure générale de recherche du minima. Le domaine $[N]$ consiste en toutes les paires de sommets (u, v). La fonction de type g retourne tout simplement l'indice de l'arbre de recouvrement qui contient u. Et la fonction à minimiser f retourne le poids de l'arête (u, v) dans le cas où l'arête existe et que u, v sont couvertes par des arbres distincts. Dans le cas contraire f retourne ∞. L'analyse de l'algorithme fournit la complexité annoncée dans le tableau 3.1.

3.2.4 Connexité dans le modèle de tableau d'adjacence

Lemme 3.5. *Étant donné un graphe $G(V, E)$ dans le modèle de tableau d'adjacence, on peut construire en $O(n)$ requêtes classiques une forêt A couvrant un ensemble de composantes connexes $\{C_1, ..., C_k\}$ pour un entier k, tel que pour chaque composant C son degrée total $m_C := \sum_{v \in C} d_v$ n'est pas plus que $|C|^2$.*

Démonstration. L'algorithme classique est le suivant :

1. Initialement l'ensemble d'arêtes A est vide.

2. Soit $S = V$ l'ensemble des sommets qui ne sont pas encore placés dans une composante.

3. Soit $k = 0$ le nombre de composantes construites jusqu'alors.

4. Tant que S est non-vide :

   a. Choisir $v \in S$ qui maximise d_v, et poser $D = \{v\}$.

   b. Parcourir les voisins w de v, et ajouter w à D, (v, w) à A jusqu'à ce (1) qu'on ait parcouru tous les voisins, ou (2) qu'on ait trouvé un voisin w qui appartient déjà à un composant C_j pour $j \leq k$.

   c. Dans le cas (1) poser $k = k + 1$ et $C_k = D$. Dans le cas (2) ajouter D à C_j. Dans les deux cas enlever D de S.

5. Afficher k, A et $C_1, ..., C_k$.

L'algorithme effectue $|A|$ requêtes au total, ce qui est $n - k$. Pour prouver qu'il est correct, soit v un sommet choisi en 4a et d son degré. Alors $d \leq |C_j|$ pour toute composante construite jusqu'alors, car la taille d'une composante à sa création est le degré d'un de ses sommets, ce qui par le choix 4a doit être au moins d et les composantes ne décroissent pas. Pour montrer que $m_C \leqslant |C|^2$ nous considérons les deux cas de 4b.

Dans le cas (1), D est le sommet v avec ses voisins, donc $m_C \leqslant d(d+1) < (d+1)^2 = |D|^2$. Dans le cas (2), soit a la taille de la composante C_j à laquelle D a été ajouté, et b la taille de D. Alors $b \leqslant d \leqslant a$. Donc le degré total est au plus $a + bd < (a + b)^2$. □

Théorème 3.6. *Étant donné un graphe $G(V, E)$ dans le modèle de tableau d'adjacence, l'algorithme quantique suivant affiche un arbre couvrant au bout d'un nombre espéré de $O(n)$ requêtes si G est connecté et ne termine jamais sinon.*

Démonstration. L'algorithme quantique est le suivant :

1. Construire A utilisant le lemme précédent.

2. Répéter jusqu'à ce que A couvre le graphe :

   a. Choisir une composante connectée de A qui minimise le degré total m_C.

   b. Chercher une arête $e \in C \times \overline{C} \cap E$ et l'ajouter à A, c'est-à-dire une arête qui connecte C à une autre composante de A. Pour cela utiliser la variante de l'algorithme de recherche quantique qui retourne une solution en temps $\Omega(\sqrt{m_C})$, s'il en existe une, et ne termine jamais sinon.

Supposons que G soit connexe, et estimons le nombre espéré de requêtes faites par l'algorithme. Nous avons d'abord calculé k composantes, chaque composante C a un degré au plus $m_C \leqslant |C|^2$. Dans chaque itération, nous choisissons la composante de degré total minimal et cherchons une arête sur son bord. Le coût espéré de cette recherche est au plus $\alpha\sqrt{m_C}$ pour une constante α. Nous distribuons ce coût uniformément sur chacun des m_C extrémités d'arêtes dans C, chaque sommet paye $\alpha/\sqrt{m_C}$.

Fixons une extrémité d'arête arbitraire v. Énumérons les au plus $\log m$ les composantes successives qui ont été choisies en 2a par l'algorithme et qui contenaient v. Soit m_i le degré total de la i-ème composante. Alors $m_{i+1} \geqslant 2m_i$. Le coût total associé à v est au plus

$$\sum_{i=0}^{\log m} \frac{\alpha}{\sqrt{m_i}} \leqslant \sum_{i=0}^{\log m} \frac{a}{\sqrt{2^i m_0}} \leqslant \frac{4\alpha}{\sqrt{m_0}}.$$

Soit C une des k composantes construites par la première étape. Le coût total associé aux extrémités d'arêtes de C est au plus $4\alpha\sqrt{m_C}$ ce qui est au plus $4\alpha|C|$. En sommant sur toutes les composantes, nous obtenons la complexité annoncée. □

3.2.5 Plus courts chemins à source unique

Une application plus subtile de l'algorithme de recherche de minima concerne la recherche des plus courts chemins à partir d'une source unique donnée. Je dis *plus subtile*, car si on applique directement l'algorithme de recherche quantique dans l'algorithme de Dijkstra, on obtient un algorithme quantique non-optimal.

Soit $G(V, E)$ un graphe avec des poids non-négatifs $w\colon E \to \mathbb{N}^\star$ sur les arêtes, et soit $d(u, v)$ la longueur du plus court chemin entre u et v. Pour plus de simplicité, on peut considérer que le graphe est complet, en le complétant avec des arêtes de poids infini. L'algorithme de Dijkstra construit pas à pas un arbre A des plus courts chemins enraciné en la source donnée v_0. Cet arbre couvre un ensemble de sommets $V(A)$ et satisfait que pour tout sommet $v \in V(A)$ le chemin dans A de v_0 à v est le plus court. Pour tout autre sommet $u \notin V(A)$, on définit une distance $\mathrm{dist}_A(u) = \min_{v\in A} d(v_0, v) + w(v, u)$, qui est la longueur du plus court chemin de v_0 à u en n'empruntant que des sommets intermédiaires de $V(A)$. Soit $\mathrm{prec}_A(u) = \mathrm{argmin}_{v\in A} d(v_0, v) + w(v, u)$, le prédécesseur de u sur ce chemin. L'algorithme de Dijkstra consiste tout simplement à choisir le sommet $u \notin V(A)$ qui minimise $\mathrm{dist}_A(u)$, et à ajouter (v, u) dans A où $v = \mathrm{prec}_A(u)$. Ceci préserve l'invariant que A est un arbre de plus courts chemins. Notre algorithme quantique diffère en la manière de trouver le sommet u à ajouter.

Dans l'algorithme de Dijkstra, deux structures de données sont maintenues, qui associent à chaque sommet u respectivement $\mathrm{dist}_A(u)$ et $\mathrm{prec}_A(u)$. A chaque itération ces structures sont mises à jour. Notre algorithme utilise une structure de donnée plus sophistiquée. L'ensemble $V(A)$ est partitionné par décomposition

binaire. Si on écrit $|V(A)|$ en binaire on décompose la cardinalité en puissances de 2. Le partitionnement de $V(A)$ contient des ensembles dont les tailles sont justement ces puissances de 2. Concrètement soit $k = \lfloor \log_2 |V(A)| \rfloor$. Alors le premier ensemble V_1 de la partition contient les 2^k premiers sommets ajoutés à A. Le reste de la partition est obtenu récursivement de $V(A) \backslash V_1$, voir figure 3.2. Soit $V_1, ..., V_\ell$ ce partitionnement, où ℓ est le nombre de termes dans la décomposition binaire de $|V(A)|$. Notre structure de données consiste alors en ensembles $U_1, ..., U_\ell$ définis comme suit. Pour tout indice $1 \leqslant a \leqslant \ell$, U_a contient les $|V_a|$ sommets $u \notin V_1 \cup ... \cup V_a$ qui minimisent $\min_{v \in V_a} d(v_0, v) + w(v, u)$.

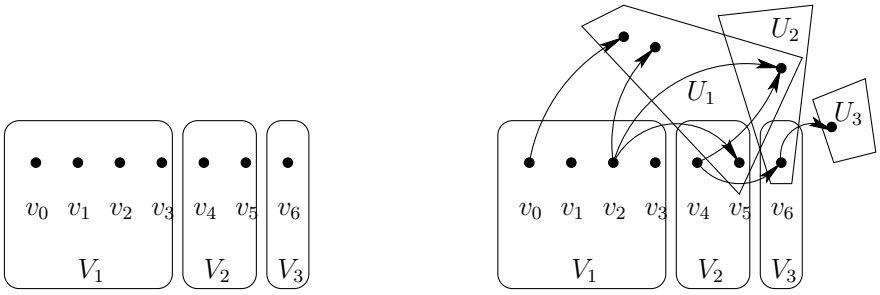


Figure 3.2. A gauche, le partitionnement de $V(A)$ pour $|V(A)| = 4+2+1$, et à droite les ensembles des voisins plus proches U_a correspondants.

Cette structure de données permet de trouver le sommet u qui minimise $\text{dist}_A(u)$, car le sommet $v = \text{prec}_A(u)$ doit appartenir à un des ensembles V_a et donc $u \in U_a$. Donc il suffit de choisir le sommet $u \in U_1 \cup ... \cup U_\ell \backslash V(A)$ qui minimise $\text{dist}_A(u)$, et de mettre à jour la structure de données. C'est-à-dire on crée un ensemble $V_{\ell+1} = \{u\}$, et on incrémente ℓ. Puis pour maintenir le partitionnement binaire, tant que les deux derniers ensembles V_{l-1} et V_ℓ ont la même taille on les fusionne. Finalement il faut calculer l'ensemble U_ℓ, pour maintenir la séquence d'ensembles U_1, ..., U_ℓ. C'est ici qu'intervient l'utilisation de l'algorithme de recherche de minima. Pour une arête (v, u) donnée avec $v \in V_\ell$ et $u \notin V_1 \cup ... \cup V_\ell$, on définit la fonction à minimiser $f(v, u) = d(v_0, v) + w(v, u)$ et le type $g(v, u) = u$. En cherchant les $|V_\ell|$ plus petites valeurs pour f qui soient g-type distincts, on trouve effectivement l'ensemble U_ℓ voulu.

Ce qui est important dans cet algorithme, c'est que la technique utilisée diffère du meilleur algorithme classique connu. Il est envisageable (et souhaitable), qu'elle puisse ensuite servir pour un algorithme classique pour chercher des plus courts chemins dans des graphes particuliers. L'informatique quantique nous force à regarder les problèmes d'une manière nouvelle et permet parfois d'obtenir de meilleurs résultats classiques. Par exemple Scott Aaronson a réussi à transformer la méthode de borne inférieure d'adversaire quantique d'Andris Ambainis pour prouver une borne inférieure classique [1]. Sophie Laplante et al. ont trouvé une méthode unifiée qui permet de prouver des bornes inférieures aux complexités en requête quantique et classique [57].

3.2.6 Problèmes de géométrie

En été 2004, avec des étudiants nous nous sommes penchés sur la question de la complexité quantique de divers problèmes en géométrie calculatoire. Nous avons identifié les problèmes importants et obtenu des résultats exacts pour certains problèmes. Jusqu'à présent, nous ne les avons pas encore publiés.

Il y avait deux travaux antérieurs sur la complexité quantique de certains problèmes en géométrie calculatoire : de Kunihiko Sadakane, Norito Sugawara, Takeshi Tokuyama [68, 69] et aussi de Warren Smith [73]. Ils traitaient de nombreux problèmes, notamment celui de calculer l'enveloppe convexe, ou de trouver un couple de points les plus proches, mais ne contenaient pas de bornes inférieures. En nous basant sur un algorithme récent de marche aléatoire quantique d'Andris Ambainis, nous avons trouvé de meilleurs algorithmes. Nous avons aussi étudié la complexité de certains problèmes sur des polygones qui n'avaient pas encore été étudiés. Dans ce contexte l'entrée est tout simplement un tableau contenant les coordonnées de tous les points, et une requête consiste à demander les coordonnées du i-ème point. Un polygone est dit *simple* s'il ne contient pas deux segments intersectant. Un polygone simple enferme alors l'intérieur du polygone, et l'ordre des points peut être pris suivant l'orientation en sens normal ou en sens des aiguilles d'une montre. Voici les différents résultats que nous avons obtenus, concernant différents problèmes de décision sur les polygones.

Problème	Quantique	Classique
Test si un polygone donné est simple	$\Omega(\sqrt{n}), O(n^{2/3})$	$\Theta(n)$
Test d'orientation d'un polygone simple	$\Theta(\sqrt{n})$	$\Theta(n)$
Test de convexité d'un polygone simple	$\Theta(\sqrt{n})$	$\Theta(n)$
Test d'appartenance d'un point à un polygone	$\Theta(n)$	$\Theta(n)$
idem pour un polygone simple	$\Theta(\sqrt{n})$	$\Theta(n)$
idem pour un polygone convexe	$\Theta(\log n)$	$\Theta(\log n)$

Tableau 3.2. Complexité de requêtes en erreur bornée pour des problèmes de décision sur des polygones. Nos résultats sont marqués en gris.

Les bornes inférieures classiques de ces problèmes découlent immédiatement des réductions suivantes. Pour prouver les bornes inférieures quantiques nous réduisons les problèmes Parité, Ou, et Recherche dans un Tableau Trié de manière très simple. Notons qu'il s'agit d'une réduction dans le modèle de requête. Ici on ne traduit pas dans un pré-calcul l'entrée du premier problème dans une entrée du deuxième problème. La réduction doit être faite de telle manière à ce que chaque requête à l'entrée du deuxième problème se traduit par un nombre constant de requêtes à l'entrée du premier problème. Pour la plupart des problèmes nous avons aussi obtenu une réduction dans le sens contraire, ce qui établit les bornes supérieures. Nous pensons que ces réductions étaient déjà connues, car elles sont très naturelles, mais nous ne les avons pas trouvées dans la littérature.

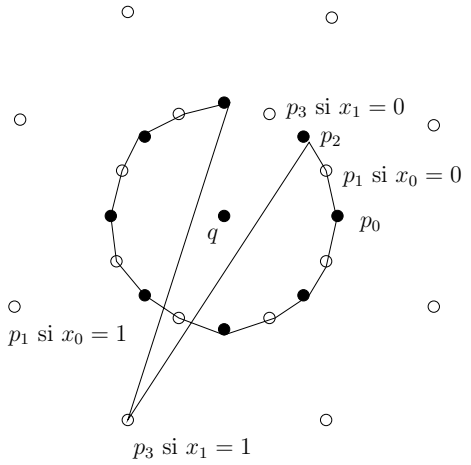


Figure 3.3. Réduction de Ou au test si un polygone est simple, et aussi de Parité au test d'appartenance d'un point q à un polygone.

Nous n'allons détailler qu'une des réductions à titre d'exemple, les autres étant tout à fait similaires. Les bornes supérieures s'obtiennent par application directe de l'algorithme de recherche quantique.

Lemme 3.7. *La complexité en requêtes quantiques à erreur bornée du test si un polygone donné est simple vaut* $\Omega(\sqrt{n})$.

Démonstration. Nous réduisons le problème Ou à ce test (voir la figure 3.3). Soit $x_0x_1...x_{n-1}$ l'entrée au problème Ou. Nous définissons le polygone P à $2n$ points comme suit. Pour tout $0 \leq i \leq n-1$, en coordonnées polaires, $p_{2i} := (\pi i/n, 1)$ et $p_{2i+1} := (\pi(i+1)/n, 1)$ si $x_i = 0$ et $p_{2i+1} := (\pi + \pi(i+1)/n, 2)$ si $x_i = 1$.

Maintenant, P est simple si et seulement si $x = 0...0$. Chaque requête à P se traduit par au plus une requête à x. Ceci établit la borne inférieure souhaitée. □

La borne supérieure est tout simplement obtenue en utilisant l'algorithme d'Andris Ambainis pour le problème d'éléments distincts et en y remplaçant le test de collision entre les indices i et j par le test d'intersection des segments $\overline{p_ip_{i+1}}$ et $\overline{p_jp_{j+1}}$. La complexité en temps est par contre bien pire que $O(n^{3/2})$ car on ne connaît pas de structure de données qui stocke un ensemble de segments et qui permette facilement de tester s'ils sont intersectés par segment donné.

3.3 Problèmes ouverts

Une série de problèmes ouverts concerne les généralisations du problème de collision. À titre d'exemple, voici un problème qui est issu de la géométrie calculatoire.

3-SOMME : Étant donné n entiers $a_1, ..., a_n$ en trouver trois a_i, a_j, a_k tel que $a_i + a_j + a_k$=0.

Ce problème peut être résolu en temps $O(n^2)$ par un algorithme classique. On conjecture que ceci est optimal et il existe une borne inférieure correspondante dans un modèle de calcul particulier, appelé arbres de décision algébriques. C'est un problème important, car il se réduit à beaucoup de problèmes géométriques [45]. Cette réduction était importante car elle donnait une explication au fait que personne n'arrivait à trouver un algorithme en $o(n^2)$ pour ces problèmes géométriques.

Les propriétés de graphe forment une famille de problèmes importants. Une propriété de graphe est une fonction booléenne, qui est invariante aux isomorphismes et qui satisfait que si G a la propriété, alors tout *sur-graphe* G' l'a aussi. Rappel : G' est sur-graphe de G si et seulement si G est sous-graphe de G' induit par les sommets. La propriété est donc entièrement définie par les graphes minimaux qui ont la propriété. Le cas le plus simple est quand il y a un seul tel graphe, ce qui est le cas par exemple avec le problème du triangle. Le problème suivant est une des propriétés de graphes les plus simples.

INCLUSION DE GRAPHE : Soit H un graphe fixé de taille constante. Étant donné un graphe G, est-ce que G contient H, c'est-à-dire est-ce qu'il existe une fonction $f: V(H) \to V(G)$ qui préserve les arêtes ?

Magniez, Santha et Szegedy [61] ont trouvé un algorithme quantique pour ce problème en $\tilde{O}(n^{2-2/k})$, où $k = |V(H)|$, mais qui semble trop brutal pour être optimal. Il n'existe pas de borne inférieure non-triviale pour ce problème.

Chapitre 4

Ordonnancement de tâches à durée égale

Ce chapitre est basé sur les articles suivants :

1. [7] *Runway scheduling with holding loop*, avec Konstantin Artiouchine et Philippe Baptiste. Dans Discrete Optimization Methods in Production and Logistics (DOM), pp. 96-101, Omsk-Irkutsk, Russia, 2004.

2. [13] *Preemptive scheduling of equal-length jobs to maximize weighted throughput*, Philippe Baptiste, Marek Chrobak, Wojciech Jawor et Nodari Vakhania. Oper. Res. Lett., 32(3):258–264, 2004.

3. [35] *A Note on Scheduling Equal-Length Jobs to Maximize Throughput*, avec Marek Chrobak, Wojciech Jawor, Lukasz Kowalik et Maciej Kurowski, A paraître dans Journal of Scheduling.

4. [14] *Preemptive Multi-Machine Scheduling of Equal-Length Jobs to Minimize the Average Flow Time*, avec Philippe Baptiste, Marek Chrobak et Francis Sourd, soumis.

4.1 Préliminaires

Essentiellement avec Marek Chrobak et Philippe Baptiste, nous avons considéré des problèmes d'ordonnancement avec durée égale des tâches et dates de relâchement. Beaucoup de problèmes d'ordonnancement issus des applications réelles sont NP-complets dans toute leur généralité. Il faut alors exploiter les particularités des problèmes. La durée unique pour toutes les tâches en est un exemple. Le

problème peut alors devenir polynomial et ceci peut parfois s'étendre aux cas où il existe un nombre constant de longueurs possibles.

Des problèmes d'ordonnancement apparaissent dans des domaines d'application très différents, comme dans l'organisation de chaînes de production ou le contrôle aérien par exemple. Chaque application apporte des contraintes différentes, des fonctions objectives différentes à optimiser. Devant cette multitude de problèmes, Graham a introduit une notation, qui est maintenant communément utilisée et qui englobe la plupart des problèmes étudiés. Chaque problème porte un nom sous la forme $\alpha|\beta|\gamma$, où α spécifie l'environnement d'ordonnancement, β décrit les contraintes imposées sur les tâches et γ est la fonction objective à minimiser. Pour donner un exemple $1|r_j; p_j = p|\sum U_j$ est le problème d'ordonnancement à une machine, où on dispose de n tâches, chacune a le même temps d'exécution p, et vient avec un intervalle $[r_j, d_j]$ dans lequel la tâche j devrait être ordonnancée de préférence. Notez que les dates limites d_j sont implicites dans cette notation. Un ordonnancement valide associe à chaque tâche j un intervalle d'exécution disjoint, de longueur p et ne débutant pas avant la date de relâchement r_j. La fonction de pénalité U_j vaut 1 si l'intervalle d'exécution termine après la date limite d_j et vaut 0 sinon. Les tâches qui terminent en retard, peuvent alors être ordonnancées arbitrairement loin, sans que cela n'augmente davantage la pénalité. Minimiser $\sum U_j$ revient alors à trouver un sous-ensemble maximal de tâches qui elles peuvent être ordonnancées toutes à temps. Cette fonction de coût est motivée par des systèmes à temps réel, où les tâches viennent avec une date limite stricte et sont en trop grand nombre pour pouvoir être toutes exécutées à temps.

Dans d'autres cas les tâches n'ont pas de date limite, et veulent juste terminer le plus vite possible. On peut alors choisir de minimiser le temps de complétude maximal des tâches ($\max C_j$ aussi appelé *makespan*) ou alors de minimiser le temps d'attente total des tâches ($\sum C_j$ aussi appelé *flowtime*). C'est cette dernière fonction de coût que nous avons considérée. Dans certains cas les tâches viennent avec des priorités différentes, on considère alors une version pondérée de la fonction de coût.

Ce sont surtout les méthode de résolution qui sont importantes, plus que les problèmes eux-mêmes. Nous espérons qu'elles vont servir à résoudre d'autres problèmes d'ordonnancement par la suite.

4.2 Contributions

Le tableau 4.1 résume une famille de problèmes avec leur complexité. Nous avons étudié les problèmes à une unique machine ou le problème plus général à m machines parallèles. Puis nous avons aussi considéré les ordonnancements sans

préemption, où les tâches doivent être ordonnancées d'une seule traite, et les ordonnancement avec préemption, où les tâches peuvent être interrompues pour être reprises plus tard, éventuellement par une autre machine. Pour la fonction de coût que nous avons considérée, la préemption rend les problèmes plus faciles dans le cas à une machine et plus difficile dans le cas à m machines parallèles.

$1\|r_j; \quad p_j = p\|\sum U_j$	$O(n^5)$ [35] corrige un algorithme de [31]
$1\|r_j; \quad p_j = p\|\sum w_j U_j$	$O(n^7)$ [10]
$1\|r_j; \text{pmtn}; p_j = p\|\sum U_j$	$O(n \log n)$ [59]
$1\|r_j; \text{pmtn}; p_j = p\|\sum w_j U_j$	$O(n^4)$ [13] améliore un algorithme en $O(n^{10})$ [10]
$P\|r_j; \quad p_j = p\|\sum U_j$	ouvert
$P\|r_j; \quad p_j = p\|\sum w_j U_j$	ouvert
$P\|r_j; \text{pmtn}; p_j = p\|\sum U_j$	NP-complet [56]
$P\|r_j; \text{pmtn}; p_j = p\|\sum w_j U_j$	NP-complet, même si $\forall j: r_j = 0$ [27]
$1\|r_j; \quad p_j = p\|\sum C_j$	$O(n \log n)$ par algo. glouton
$1\|r_j; \quad p_j = p\|\sum w_j C_j$	$\in P$ [11]
$1\|r_j; \text{pmtn}; p_j = p\|\sum C_j$	$O(n \log n)$ [12] observe que la préemption est inutile ici
$1\|r_j; \text{pmtn}; p_j = p\|\sum w_j C_j$	ouvert
$P\|r_j; \quad p_j = p\|\sum C_j$	$O(n \log n)$ par algo. glouton
$P\|r_j; \quad p_j = p\|\sum w_j C_j$	ouvert
$P\|r_j; \text{pmtn}; p_j = p\|\sum C_j$	progr. lin. de taille $O(nm)$ [14], améliore $O(n^3)$ [28]
$P\|r_j; \text{pmtn}; p_j = p\|\sum w_j C_j$	unairement NP-complet [60]

Tableau 4.1. Complexités de certains problèmes d'ordonnancement. Nos contributions sont marquées.

4.2.1 Par programmation dynamique

Beaucoup de problèmes NP-complets peuvent être résolus en temps polynomial par la programmation dynamique à partir du moment où les tâches ont toutes la même durée. Ceci est dû au fait que dans ce cas on peut souvent considérer que les ordonnancements optimaux ont la forme particulière où tous les temps intéressants (début, interruption, reprise ou fin d'une tâche) sont de la forme $r_j + ap$ où r_j est la date de relâchement d'une tâche, p la durée unique des tâches, et a un entier entre 0 et n. On peut alors décomposer l'ordonnancement en intervalles, dont les limites sont prises d'un ensemble de taille $O(n^2)$ et en dégager un programme dynamique qui calcule un ordonnancement optimal. Cette approche est une idée

générale qui doit cependant être adaptée à chaque problème de manière différente. Examinons maintenant les problèmes où nous avons pu l'appliquer.

4.2.1.1 Minimiser le nombre de tâches en retard sur une machine

Nous décrivons maintenant notre contribution au problème d'ordonnancement sur une machine, avec durée égale, dates de relâchement et minimisant le nombre de tâches qui terminent en retard. Ce problème porte le nom $1|r_j; p_j = p| \sum U_j$.

La fonction objective la plus étudiée est $C_{\max}$, définie par $\max C_j$ aussi appelée *makespan* en anglais. Dans ce problème les tâches n'ont pas de date limite, il s'agit juste de calculer un ordonnancement qui minimise le temps de la fin globale de l'ordonnancement. Ce problème a été très étudié, et résolu indépendamment par Jacques Carlier d'une part [31] et Barbara Simons d'autre part [72] et finalement résolu en temps $O(n \log n)$ par Garey, Johnson, Simons, et Tarjan [48]. Ce qui est intéressant c'est que les deux premiers algorithmes sont complètement différents. L'algorithme de Simons résout en fait un problème plus général où chaque tâche vient avec une date limite stricte, qui est notée $\bar{d_j}$ et le but est de trouver un ordonnancement faisable, c'est-à-dire qui respecte les dates de relâchement et les dates de limite. L'algorithme calcule des zones interdites dans lesquelles une tâche ne peut pas commencer sans que plus tard une date limite soit violée, puis ordonnance de manière gloutonne les tâches en dehors de ces zones. L'amélioration en $O(n \log n)$ repose sur un calcul subtil de ces zones qui fait intervenir une structure de donnée digne de Tarjan. Par contre l'algorithme de Carlier repose sur la programmation dynamique.

Dans le même article il généralise son algorithme au problème de minimiser le nombre de tâches en retard, qui était un problème encore ouvert à cette époque. Nous nous sommes rendus compte avec Marek Chrobak et Wojciech Jawor que cette généralisation comporte une erreur subtile, et que non seulement l'algorithme ne se corrige pas, mais que l'approche utilisée ne peut pas aboutir. Plus précisément, Carlier utilisait un programme dynamique, où pour chaque temps $t \leq t_0$ il avait calculé un ensemble d'ordonnancements partiels $\mathcal{S}_t$, qui sont encore vides après t. Puis l'ensemble $\mathcal{S}_{t_1}$ pour un temps $t_1 > t_0$ était constitué d'ordonnancements de $\mathcal{S}_t$ pour $t \leq t_0$ éventuellement complétés à la fin par l'exécution d'une tâche supplémentaire. Ce calcul se faisait uniquement en comparant les dates limites entre elles et avec t_1. L'invariant était bien sûr que pour tout temps t, $\mathcal{S}_t$ comporte un ordonnancement qui peut être complété vers un ordonnancement optimal. Nous avons construit une instance du problème avec quatre tâches où un tel algorithme devrait stocker deux ordonnancements partiels à un moment donnée pour être correct. Nous avons ensuite amplifié cette construction à une instance avec $4n$ tâches pour laquelle tout algorithme correct devrait stocker à un moment donné 2^n ordonnancements partiels. Ceci mettait fin aux tentatives des corriger

l'algorithme par programmation dynamique de [31].

Par contre on savait que ce problème pouvait être résolu en temps polynomial, car il existe un algorithme par programmation dynamique en temps $O(n^7)$ pour le problème plus général, qui consiste à minimiser le nombre *pondéré* de tâches en retard $(1|r_j; p_j = p| \sum w_j U_j)$ [10]. Avec une approche similaire, mais utilisant le fait que les tâches n'ont pas de poids, Lukasz Kowalik et Maciej Kurowski ont trouvé un algorithme en $O(n^5)$ pour ce problème [35].

4.2.1.2 Minimiser avec préemption le poids total des tâches en retard

Nous décrivons maintenant notre contribution sur le problème d'ordonnancement sur une machine, à durée égale, avec dates de relâchement et une pondération sur les tâches, minimisant le poids total des tâches qui terminent en retard. Ce problème porte le nom $1|r_j; \text{pmtn}; p_j = p| \sum w_j U_j$.

Dans certains cas on veut un ordonnancement, qui n'affecte pas forcément à toutes les tâches un unique intervalle d'exécution de longueur p, mais une suite d'intervalles dont la longueur totale vaut p. Il s'agit alors d'un ordonnancement préemptif. Le cas non-pondéré a été résolu par Lawler avec un algorithme extrêmement élégant [59] en temps $O(n \log n)$. Le cas pondéré par contre a été résolu seulement récemment par Philippe Baptiste avec la programmation dynamique en temps $O(n^{10})$. Avec Marek Chrobak, Wojciech Jawor, Nodari Vakhania et Philippe Baptiste nous avons réussi à trouver un meilleur algorithme dont la complexité n'est que $O(n^4)$. Il utilise deux propriétés très simples sur les ordonnancements. Supposons que les tâches soient numérotées dans l'ordre des dates limites $d_1 \leq ... \leq d_n$, et soit S un ordonnancement faisable, c'est-à-dire où chaque tâche ne commence pas avant sa date de relâchement et ne termine pas après sa date limite. Alors est également faisable l'ordonnancement EDD — pour *earliest due date* — qui exécute à tout moment la tâche la plus urgente parmi les tâches en suspension, c'est-à-dire qui sont déjà relâchées et pas encore complétées. Un tel ordonnancement a une structure récursive : si on y supprime la tâche avec la date limite la plus grande, on obtient l'ordonnancement EDD pour les tâches restantes. De plus l'ordonnancement consiste en blocs séparés par des temps morts. Un bloc est un intervalle $[r_i, C_j)$ entre une date de relâchement r_i et une date de complétude C_j et satisfait (a) que toutes les tâches exécutées dans cet intervalle ont été relâchées dans ce même intervalle et (b) que C_j est le premier temps où toutes les tâches commencées sont aussi complétées. Alors un ordonnancement EDD satisfait que si on retire la tâche la moins urgente d'un bloc, celui sera décomposé en plusieurs blocs plus petits. En même temps la condition (a) garantie que l'ensemble des tâches du bloc initial est décomposé et associé de manière unique à chacun des nouveaux blocs. Ces observations mènent vers un programme dynamique dont le temps de calcul est $O(n^4)$, que nous exposons maintenant.

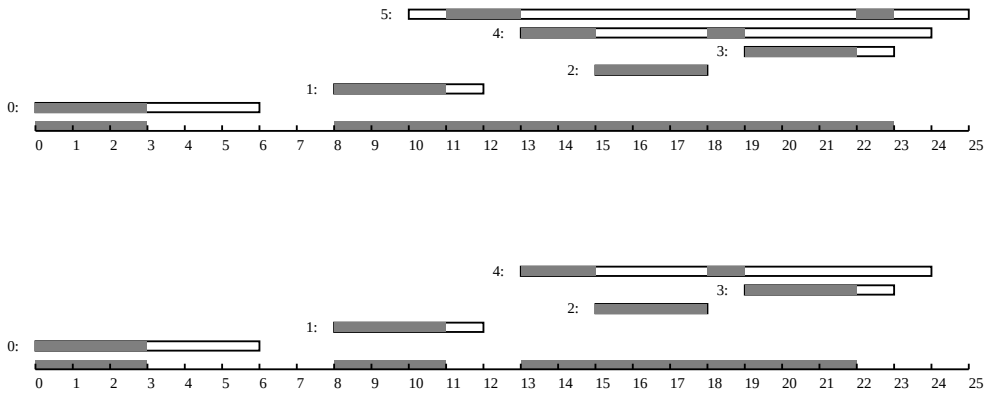


Figure 4.1. La structure récursive d'un ordonnancement EDD. Les tâches sont représentés par des rectangles étirés de la date de relâchement à la date limite. L'ordonnancement est indiqué par les zones grises. Nous les avons projetées sur l'axe de temps afin de visualiser les blocs d'ordonnancement.

On suppose que les tâches sont numérotées dans l'ordre de la date limite $d_1 \leqslant ... \leqslant d_n$ pour simplifier les notations. Avant d'introduire le programme dynamique nous devons poser quelques définitions.

Étant donné un intervalle $[x, y)$ on dit qu'un ensemble de tâches $\mathcal{J}$ est (k, x, y)-faisable si

(f1). $\mathcal{J} \subseteq \{1, ..., k\}$,

(f2). pour tout $j \in \mathcal{J}$, $r_j \in [x, y)$ et

(f3). $\mathcal{J}$ admet un ordonnancement faisable en $[x, y)$, c'est-à-dire à y toutes les tâches $j \in \mathcal{J}$ sont complétées.

Sans perte de généralité, supposons que la dernière tâche satisfasse $r_n = d_{n-1}$ et $w_n = 0$. Alors notre but est de trouver un ensemble $\mathcal{J}$ de poids total maximal qui soit $(n, 0, n)$-faisable. Pour cela nous définissons les variables

F_{ij}^k = le poids optimal d'un ensemble (k, r_i, r_j)-faisable,

G_{ia}^k = le poids optimal d'un ensemble $(k, r_i, r_i + ap)$-faisable qui consiste en un seul bloc couvrant $[r_i, r_i + ap)$,

H_{ij}^k = le poids optimal d'un ensemble (k, r_i, r_j)-faisable qui est sans temps mort entre r_i et r_{k+1}. Cette variable est définie seulement pour $r_i \leqslant r_{k+1} \leqslant r_j$.

Notre but est donc de calculer F_{1n}^n. Notez qu'il est facile de transformer le programme linéaire pour qu'il calcule en fait l'ensemble optimal de tâches, plutôt que son poids. Avant de donner une définition récursive de ces variables, il nous

faut introduire les fonctions utiles suivantes.

$$\begin{aligned}\Delta(x,y) &= \min\{n, \left\lfloor \frac{y-x}{p} \right\rfloor\}\\ \eta(x) &= \operatorname{argmin}_i\{r_i : r_i > x\}\\ \lambda(x) &= \operatorname{argmin}_i\{r_i : r_i \geqslant x\}\end{aligned}$$

Donc $\Delta(x,y)$ est le nombre maximal de tâches qui peuvent tenir dans l'intervalle $[x,y)$ sans prendre en considération les contraintes sur les dates. Puis $\eta(x)$ et $\lambda(x)$ sont la prochaine date de relâchement après x, incluant x pour λ.

Valeurs F. si $r_j \leqslant r_i$ alors $F^k_{ij} = 0$. Sinon

$$F^k_{ij} = \max \begin{cases} F_{\eta(r_i),j} & (F1)\\ \max\limits_{\substack{1\leqslant a\leqslant n\\ r_i+ap\leqslant r_j}} G^k_{ia} + F^k_{\lambda(r_{i+ap}),j}. & (F2)\end{cases}$$

Valeurs G. Si $k=0$ ou $a=0$ alors $G^k_{ia}=0$. Si $r_k \notin [r_i, r_i+(a-1)p]$ ou $d_k < r_i + a\,p$ alors $G^k_{ia} = G^{k-1}_{ia}$. Sinon

$$G^k_{ia} = \begin{cases} G^{k-1}_{ia} & (G1)\\ G^{k-1}_{i,a-1} + w_k & (G2)\\ \max\limits_{\substack{\max\{r_k,r_i\}<r_\ell<r_i+ap\\ r_\ell - r_i \notin p\mathbb{N}}} \{H^{k-1}_{i\ell} + G^{k-1}_{\ell,\Delta(r_\ell,r_i+ap)} + w_k\}. & (G3)\end{cases}$$

Valeurs H. De nouveau si $k=0$ ou $r_j \leqslant r_i$ alors $H^k_{ij}=0$. Si $k=n$ ou que $r_{k+1} \notin [r_i, r_j]$ alors H^k_{ij} n'est pas défini. Sinon

$$H^k_{ij} = \max_{\substack{0\leqslant a<n\\ r_{k+1}\leqslant r_i+ap\leqslant r_j}} \{G^k_{ia} + F^k_{\lambda(r_i+ap),j}\}. \qquad (H)$$

La preuve de correctude de ce programme linéaire est une étude de cas. Soit S un ordonnancement optimal qui est (k,i,j)-faisable. Alors soit S débute en r_i par un temps mort, ce qui est capturé par (F1), soit il débute avec un bloc d'un certain nombre de tâches, ce qui est capturé par (F2).

Maintenant soit S un ordonnancement $(k, i, r_i + ap)$-faisable qui consiste en un seul bloc de r_i à $r_i + ap$. Soit S ne contient pas la tâche k, ce qui est capturé par (G1), soit S contient la tâche k. Dans ce cas, comme c'est la tâche la moins urgente dans S, elle s'exécute soit en dernier ce qui est capturé par (G2), soit dans les temps morts laissés par les autres tâches. Dans ce cas $S\backslash\{k\}$ consiste en une séquence d'au moins deux blocs, dont le premier temps mort n'est pas avant r_k. Comme tout bloc commence à une date de relâchement, le dernier bloc de cette séquence doit commencer à une certaine date r_ℓ. Ceci est capturé par (G3).

Finalement soit S un ordonnancement (k,i,j)-faisable qui est sans temps mort avant r_{k+1}. Alors S commence par un bloc B avec un certain nombre de tâches a. Le reste de S est un ordonnancement (k, ℓ, j)-faisable, où ℓ est la première date de relâchement après la fin de B. Ceci est capturé par (H).

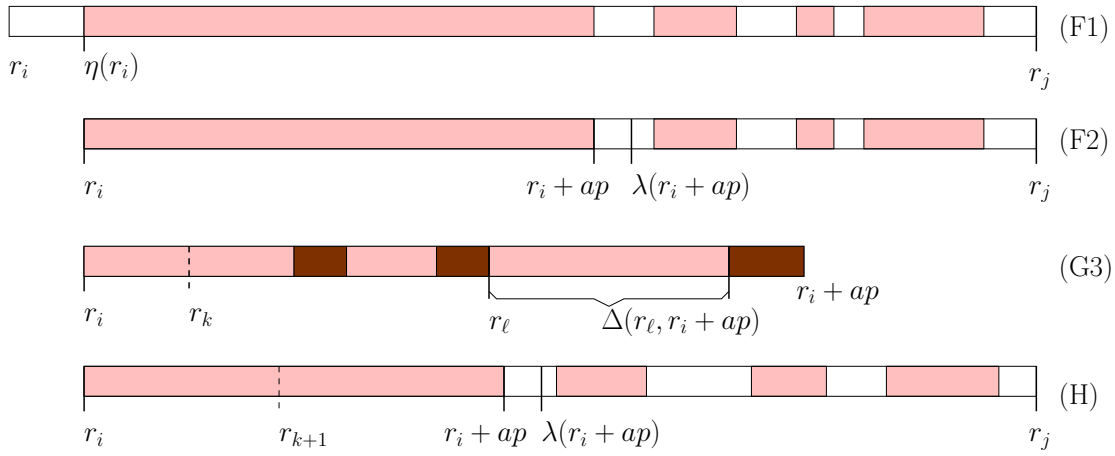


Figure 4.2. Illustration de la définition des variables $F_{ij}^k, G_{ia}^k, H_{ij}^k$ et de l'idée de la preuve.

4.2.1.3 Un problème d'ordonnancement issu du contrôle aérien

Un tout autre problème motivé par une application concrète est le suivant. Considérons un aéroport avec une unique piste d'atterrissage, et des avions qui arrivent à des moments différents de la journée pour atterrir. La distance entre deux atterrissages doit être au moins p et cette constante ne dépend pas des avions, car on suppose que tous les avions sont du même type. Cette distance nécessaire est due au temps qu'il faut pour que les turbulences générées par un atterrissage disparaissent. Pour faire respecter cette contrainte nous avons la possibilité de faire retarder ou accélérer l'atterrissage dans une certaine mesure une fois que l'avion est entré dans la zone d'atterrissage. Si jamais cette liberté n'est pas suffisante, il y a la possibilité de faire patienter l'avion dans une boucle d'attente.

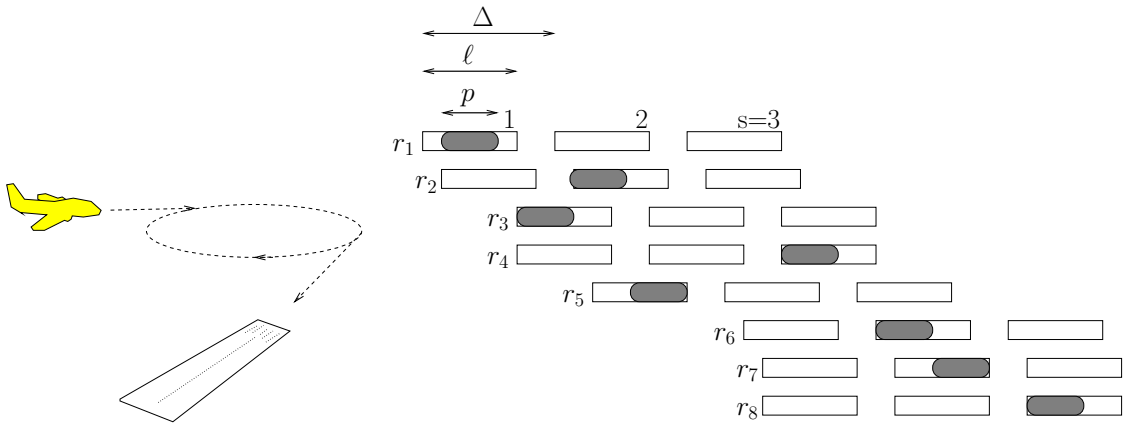


Figure 4.3. Motivation du problème de tarmac : chaque passage dans la boucle d'attente dure T unités, l'avion peut effectuer au plus $s-1$ passages, ce qui résulte en s fenêtres d'atterrissage autorisées régulières.

Dans ce problème il y a plusieurs valeurs qu'on peut vouloir optimiser. Par exemple on peut vouloir minimiser le nombre maximal de passage dans la boucle d'attente par avion pour minimiser les coûts de carburant, ou alors on peut vouloir maximiser la distance minimale p entre deux atterrissages pour maximiser la sécurité. Pour répondre à ces questions on étudie le problème de décision suivant.

PROBLÈME DE TARMAC

entrée. temps $r_1, ..., r_n$, durées p, ℓ, Δ nombre s

interprétation. il y a n tâches, chaque tâche a une durée p et doit être exécutée dans un des intervalles $[r_j + i\Delta, r_j + i\Delta + \ell)$ pour $0 \leq i < s$.

sortie. un ordonnancement pour une machine qui satisfait ces contraintes.

La complexité de ce problème est ouvert à ce jour. Il a été introduit avec un formalisme différent par Bayen et Tomlin, qui ont donné un programme linéaire à variables réelles et entières (appelé *MIP* pour mixed integer programming) ce qui ne permet pas de résoudre le problème en temps polynomial dans le pire des cas [18]. Philippe Baptiste et Konstantin ont trouvé un autre programme MIP qui utilise davantage les propriétés structurelles du problème et donne de meilleurs résultats expérimentaux [7].

En ce qui concerne la complexité exacte, des cas particuliers ont été montrés polynomiaux et des cas plus généraux ont été montrés NP-complets. Concrètement si $p = \ell$ alors on est devant un problème d'assignation de tâches à des fenêtres, tel que les fenêtres choisies soient disjointes. Ce problème a été étudié dans le cadre plus général où chaque tâche dispose d'un ensemble de fenêtres, sans qu'elles soient nécessairement à distance régulière de Δ. Pour $s = 2$ fenêtres, ce problème est clairement une instance de 2-SAT et peut donc être résolu en temps linéaire. Ceci avait été observé par [51]. En revanche pour $s = 3$ fenêtres ce problème est NP-complet [74].

Mais retournons à notre problème où les fenêtres sont disposées de manière régulière. Nous avons trouvé une solution par programmation dynamique qui résout en temps $O(n^{\Delta+1}s)$ le problème pour le cas où $p = \ell$. Nous avons procédé comme suit. Soit $r_j \bmod T$ le *type* d'une tâche. Si deux tâches de même type ont des fenêtres qui intersectent, alors les fenêtres qui intersectent sont identiques. On peut alors échanger l'exécution des tâches pour que les tâches d'un même type soient ordonnancées dans l'ordre de leur dates de relâchement. La programmation dynamique balaye l'axe de temps t, et pour tout vecteur $(k_0, ..., k_{\Delta-1})$ décide s'il existe un ordonnancement qui exécute avant le temps t les k_a premières tâches pour tous les types $0 \leqslant a \leqslant \Delta - 1$.

Une autre approche donne un programme dynamique en temps — je suis désolé pour la longueur de la formule — $O(n^2 s^3 \Delta^2 p^{-2} 2^{2s\Delta/p})$, qui pour $s\Delta/p$ fixé est le polynôme $O(sn^2)$. Considérons un ordonnancement où chaque tâche débute le plus tôt possible (immédiatement après la fin d'une autre tâche ou en début d'une fenêtre. Dans un tel ordonnancement les dates d'exécution seront prises d'un ensemble de taille $O(sn^2)$

$$\Theta := \{r_{ij} + ap : i \in \{1, ..., n\}, a \in \{0, ..., n-1\}, j \in \{1, ..., s\}\}.$$

Puis brutalement on calcule pour tout temps $t \in \Theta$ un ensemble d'ordonnancements partiels $\mathcal{S}_t$ qui sont encore vides après t et qui contiennent toutes les tâches devant nécessairement terminer avant t. Ce calcul se fait en complétant certains ordonnancements de $\mathcal{S}_{t'}$ pour des temps $t' \in \Theta$, $t' < t$. Puis on observe qu'à tout moment il y a au plus $2s\Delta/p$ tâches déjà relâchées mais pas encore expirées, d'où $|\mathcal{S}_t| \leq 2^{2s\Delta/p}$. Ceci mène naturellement, et sans grande difficulté à un programme dynamique avec la complexité indiquée ci-haut.

4.2.2 Par programmation linéaire

Une autre technique souvent utilisée en ordonnancement est la programmation linéaire. La plupart du temps il est possible de décrire un problème d'ordonnancement par un système de contraintes linéaires à variables entières. En général ce type de système ne peut pas être résolu en temps polynomial dans le pire des cas. Pour certains problèmes réels ceci est acceptable, car on veut à tout prix trouver un bon ordonnancement, et pour de toutes petites entrées un algorithme surpolynomial peut être acceptable. Par contre il y a des problèmes pour lesquels on peut se passer des contraintes entières, et le programme linéaire peut alors être résolu en temps polynomial.

Nous décrivons maintenant un tel programme linéaire que nous avons trouvé pour minimiser le flot total sur des machines parallèles, ou plus précisément pour le problème $P|r_j; \text{pmtn}; p_j = p| \sum C_j$. Dans ce problème on dispose de m machines parallèles et de n tâches, chacune est relâchée à une date r_j avant laquelle elle ne peut pas être ordonnancée. Il s'agit de trouver un ordonnancement préemptif qui minimise la somme des temps de complétude des tâches, ou de manière équivalente le temps d'attente total pour toutes les tâches. Ce problème a été résolu en 2004, par Brucker et Kravchenko [28]. Leur solution consiste en un programme linéaire de taille $O(n^3)$ suivi d'un post-traitement compliqué pour extraire un ordonnancement à partir de la solution du programme linéaire.

Avec Marek Chrobak, Philippe Baptiste et Francis Sourd nous avons trouvé une propriété structurelle qui mène vers un programme linéaire très simple de taille n m et dont la solution décrit directement un ordonnancement optimal [14]. Cette propriété est basée sur une *opération de réduction*. Soit S un ordonnancement optimal et i, j deux tâches avec $r_i < r_j$. Soit I l'ensemble des intervalles de temps où exactement une des deux tâches est exécutée. Soit t le médian de I, c'est-à-dire

$|[0,t] \cap I| = \frac{1}{2}|I|$. Alors la réduction de i et j consiste à produire un ordonnancement S' identique à S, excepté dans I, où avant t toute exécution de j est remplacée par i et après t toute exécution de i est remplacé par j. On peut montrer que cette opération n'augmente pas la fonction objective et respecte les durées des tâches ainsi que leurs dates de relâchement. Notre contribution principale a été de montrer qu'il existe un ordonnancement optimal irréductible, c'est-à-dire où toute réduction entre deux tâches i, j serait sans effet. Un tel ordonnancement a une forme très particulière, comme illustrée en figure 4.4.

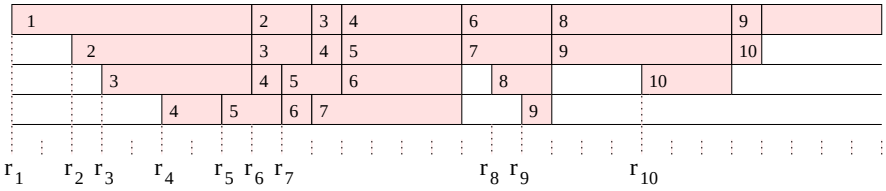


Figure 4.4. Un ordonnancement irréductible

Dans un ordonnancement irréductible chaque tâche j s'exécute sur chaque machine q dans au plus un intervalle $[S_{jq}, C_{jq})$, éventuellement vide et on peut supposer que ces intervalles sont ordonnés par machine : $C_{jq} \leq S_{j,q-1}$ pour toute tâche j et machine $q > 1$ et ordonnés par tâches $C_{jq} \leq S_{j+1,q}$ pour toute tâche $j < n$ et machine q. Comme on peut montrer que la date de complétude d'une tâche j est C_{j1}, le programme linéaire suivant trouve un ordonnancement optimal.

$$
\begin{array}{lllll}
\text{minimise} & \sum_{j=1}^{n} C_{j,1} & & & \\
\text{en respectant} & -S_{j,m} & \leq -r_j & j = 1,...,n & \\
& \sum_q (C_{j,q} - S_{j,q}) & = p & j = 1,...,n & \\
& S_{j,q} - C_{j,q} & \leq 0 & j = 1,...,n,\ q = 1,...,m & \\
& C_{j,q} - S_{j,q-1} & \leq 0 & j = 1,...,n,\ q = 2,...,m & \\
& C_{j,q} - S_{j+1,q} & \leq 0 & j = 1,...,n-1,\ q = 1,...,m &
\end{array}
$$

4.3 Problèmes ouverts

Nous avons utilisé la programmation dynamique et la programmation linéaire pour résoudre des problèmes d'ordonnancement. Mais pour pouvoir les appliquer il fallait prouver des propriétés structurelles sur les ordonnancements optimaux. C'est ce dernier point qui est, à mon avis, la contribution principale qui mène à un algorithme polynomial. Il est alors important de généraliser ces transformations structurelles afin de les appliquer à d'autres problèmes encore ouverts.

Récemment Svetlana Kravchenko et Peter Brucker ont résolu en temps polynomial une série de problèmes de la manière suivante [29]. D'abord ils décrivent les problèmes par un programme linéaire à variables entières. Ensuite ils montrent que le programme linéaire relâché (sans les contraintes entières) admet toujours une solution entière. Ce résultat surprenant mène alors vers un algorithme polynomial. Je pense qu'il devrait être possible d'en extraire une propriété structurelle des ordonnancements optimaux, qui permettrait un algorithme polynomial direct et plus simple. Une meilleure compréhension mène souvent vers la solution d'autres problèmes liés, espérons que ça sera le cas. Des problèmes ouverts sont mentionnés

dans le tableau 4.1.

Chapitre 5

Perspectives

Les problèmes élémentaires qui restent ouverts en tomographie discrète sont très difficiles, dans le sens où beaucoup d'approches ont échoué pour montrer qu'ils sont dans P ou pour les classer NP-complet. Je pense qu'il faudrait maintenant étudier dans ce domaine les problèmes réels. Pour cela il faudrait travailler en étroite collaboration avec des centres de tomographie dans les hôpitaux et des sections de contrôle de qualité non-destructif en industrie. Dans ce contexte il faudrait utiliser davantage ce qui a été fait en tomographie continue, car certaines techniques pourraient trouver un équivalent discret.

Dans un tout autre registre, en calcul quantique, on est toujours à la recherche de nouveaux algorithmes quantiques qui donneraient une accélération exponentielle par rapport au monde classique. L'algorithme de factorisation ou les algorithmes quantiques pour des problèmes sur des groupes sont basés sur un unique outil quantique, l'échantillonnage de Fourier quantique. Pendant un moment on se posait alors la question de savoir si un ordinateur quantique ne serait pas tout simplement un ordinateur classique muni d'une boîte noire d'échantillonnage de Fourier. Récemment Farhi et al. ont défini un problème artificiel qui peut être résolu par un ordinateur quantique dans un temps exponentiellement plus petit qu'un ordinateur classique [6]. Cette fois-ci, l'ingrédient quantique est une marche aléatoire quantique dont les probabilités d'observation se concentrent sur le sommet *opposé* au sommet de départ d'un graphe. Cette notion *d'opposé* n'est pas encore bien comprise pour des graphes qui diffèrent de la construction de Farhi et al. Il serait souhaitable d'explorer d'avantage le pouvoir de calcul qui résulte de l'utilisation des marches quantiques. Dans ce contexte, un projet qui me tient à coeur est l'utilisation algorithmique des effets physiques. Par exemple la lumière emprunte toujours le plus court chemin entre deux points.

Une toute autre approche consiste à chercher des algorithmes quantiques améliorant des algorithmes classiques déjà polynomiaux. Je me suis intéressé à la complexité en requêtes des propriétés de graphes, et maintenant il faudrait s'attaquer

à d'autres problèmes courants, comme des problèmes de flots ou des problèmes d'ordonnancement. La nature du calcul quantique nous invite à regarder des problèmes qui présentent une forte symétrie.

Par contre si on recherche une amélioration plus que polynomiale, on peut se demander quels sont les problèmes candidats. On pense que les problèmes NP-complets resteront difficile en quantique. Il faut donc chercher parmi les quelques problèmes entre P (ou BPP) et NP-dur. En particulier il existe cette classe de complexité intéressante TFNP, qui contient les fonctions totales (définies sur tout le domaine) et pour lesquelles il est facile de vérifier qu'une chaîne est bien l'image d'une autre. Pour être exact il faudrait remplacer fonction totale par relation R dans cette définition, avec la condition que pour tout x il existe un y qui satisfait $R(x, y)$. Pour une relation donnée il y a plusieurs manières de prouver cette existence, par exemple en utilisant une fonction de potentiel ou en utilisant le fait que dans tout graphe acyclique dirigé il existe un puits, ou par le principe des tiroirs (pigeonhole principle) ou par le fait que tout graphe de degré maximal 2 comporte un nombre pair de feuilles. Ces manières sous-divisent TFNP en PLS, PPP, PPA, etc. Récemment Christos Padamitriou a exhibé une classe de jeux pour lesquels le problème consistant à de trouver un équilibre de Nash pur est PLS-complet [41]. En pratique ces problèmes trouvent leur application par exemple dans les réseaux. Quand le routage se fait de manière non-centralisée, le comportement égoïste des entités communicantes engendre une congestion plus importante, c'est le prix de l'anarchie. La classe TFNP contient donc des problèmes, qui sont importants aussi en dehors du calcul quantique. Marios Mavronicolas a posé le problème de trouver un équilibre de Nash pur, dans le jeu où il existe m liens entre deux machines et où n utilisateurs veulent ouvrir une connexion entre ces machines. L'entrée du problème est une matrice $n \times m$ qui détermine quelle sera la contribution à la congestion du lien j si l'utilisateur i ouvre sa connexion sur ce lien. La congestion d'un lien est alors la somme des contributions de ses utilisateurs. Le but étant de trouver une assignation des utilisateurs aux liens, telle que qu'aucun utilisateur n'a intérêt à changer de lien. Je cherche un algorithme polynomial pour ce problème, et je l'ai déjà résolu pour $m = 2$.

A long terme je me propose d'étudier plus en détail la complexité d'une part du problème qui consiste à trouver un équilibre de Nash pur pour différents jeux, et d'autre part du problème de conception, où il s'agit de trouver un jeu pour lequel les équilibres satisfont des propriétés données. Dans ce contexte, il faudrait produire de nouveaux algorithmes d'une part, et de nouvelles réductions d'autre part pour montrer la PLS-complétude de certains problèmes particulièrement difficiles. Nous avons mentionné que ces problèmes sont motivés par l'analyse de la croissance d'un réseau par des opérateurs de télécommunication égoïstes. Dans ces contextes, les demandes de ressources supplémentaires arrivent en-ligne. J'étudierai alors dans quels cas des algorithmes en-ligne peuvent produire un équilibre de Nash pur. L'algorithmique en-ligne est traditionnellement étudié pour

la gestion de caches et pour des problèmes d'ordonnancement, ce qui nous amène à la dernière partie de ce chapitre :

Ce que je considère comme important en informatique, et aussi extrêmement passionnant, c'est de savoir distinguer dans une famille de problèmes lesquels sont difficiles et lesquels faciles. L'ordonnancement consiste à attribuer des plages horaires à des tâches données, en respectant des contraintes de date limite, de dépendance entre les tâches, etc, afin de minimiser une fonction de coût donnée. En fonction du modèle de machine, des contraintes sur les tâches et de la fonction objective il existe un nombre incroyable de problèmes. Certains sont ouverts, certains NP-complet, certains polynomiaux. Une liste de Brucker et Knust recense ces problèmes. J'aimerais m'attaquer à des problèmes ouverts, puis de manière moins ambitieuse je veux trouver de meilleurs algorithmes pour certains cas polynomiaux connus. Des problèmes d'ordonnancement apparaissent dans de nombreuses situations, comme par exemple le contrôle aérien, la planification d'une chaîne de production où dans les systèmes d'exploitation. Nous voyons donc que c'est un domaine important dans lequel il est nécessaire de développer différentes techniques algorithmiques et de distinguer ce qui rend certains problèmes tellement difficiles à résoudre. Je pense qu'il est aussi important d'étudier des problèmes concrets, comme par exemple ceux issus du contrôle aérien ou ferroviaire, et de fournir des approximations pour les cas difficiles.

Dans ce contexte, je me propose d'étudier l'algorithmique en-ligne. Pour donner un exemple concret, le problème 2-serveur consiste en un disque dur avec deux têtes de lecture/écriture indépendantes, et des requêtes de lecture/écriture arrivent pour des cylindres sur ce disque. Il s'agit alors de décider laquelle des têtes il faut déplacer pour répondre à la requête afin de minimiser le temps de déplacement total. Un algorithme est α-compétitif si le coût est au plus α fois le coût optimal si on avait su à l'avance la séquence des requêtes. Ceci peut être vu comme un jeu répété à l'infini où le premier joueur est l'algorithme en-ligne et l'autre l'adversaire qui génère les requêtes. Depuis quelque temps je travaille sur des problèmes d'ordonnancement et je vais étendre mon travail sur l'ordonnancement en-ligne.

Voici deux des problèmes auxquels je vais m'attaquer dans les deux, trois années qui suivent. Je m'intéresse particulièrement aux problèmes d'ordonnancement à une machine où chaque tâche vient avec un intervalle d'ordonnancement autorisé borné par une date de relâchement et une date limite. Le but est de trouver un ordonnancement qui maximise le nombre total de tâches qui sont ordonnancées dans leur intervalle. Ce problème s'écrit $1|r_j|\sum U_j$ dans la notation de Graham, et est NP-complet dans le sens fort. Suivant qu'on autorise la préemption ou pas, que les tâches ont la même durée ou pas, que les tâches ont un poids ou pas, etc, il existe des variantes de ce problème. Comme souvent en ordonnancement, ces problèmes ne sont pas tous motivés par des applications dans la vie réelle, mais on espère que les techniques développées seront utiles.

Pour des tâches à durée égale et pondérées, le problème $1|r_j; p_j = p|\sum w_j U_j$ peut être résolu en temps $O(n^7)$, ce qui est certes polynomial, mais trop important en pratique. Nous travaillons alors sur un nouvel algorithme qui serait plus efficace, en découpant l'ordonnancement en blocs, et en donnons une solution différente pour les blocs de petite taille et les blocs de grande taille.

Considérons maintenant le même problème pour des durées unitaires $1|r_j; p_j = 1|\sum w_j U_j$ avec la restriction que les temps de relâchement et dates limites sont entiers. Il apparaît dans les routeurs de réseaux synchrones. Alors que le cas hors-ligne est extrêmement simple — c'est un problème de couplage — le cas en-ligne est compliqué. L'algorithme glouton par exemple est 2-compétitif. Mais actuellement les meilleurs bornes inférieures sur le rapport sont le nombre d'or 1.618... pour le cas déterministe et $e/(e-1) = 1.58...$ pour le cas probabiliste. Beaucoup d'efforts ont été fournis pour améliorer ces bornes, en ajoutant des contraintes supplémentaires. C'est aussi ce que je me propose de réaliser.

Bibliographie

[1] Scott Aaronson. Lower bounds for local search by quantum arguments. In *Proceedings of the thirty-sixth annual ACM Symposium on Theory of Computing (STOC)*, pages 465–474, 2004.

[2] Scott Aaronson and Yaoyun Shi. Quantum lower bounds for the collision and the element distinctness problems. *Journal of the ACM*, 51(4):595–605, 2004.

[3] Andris Ambainis. Quantum lower bounds by quantum arguments. *J. Comput. System Sci.*, 64(4):750–767, 2002. Special issue on STOC 2000 (Portland, OR).

[4] Andris Ambainis. Polynomial degree vs. quantum query complexity. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 230–239, 2003.

[5] Andris Ambainis. Quantum walk algorithm for element distinctness. In *Proc. of the Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 22–31, 2004.

[6] Enrico Deotto Edward Farhi Sam Gutmann Andrew M. Childs, Richard Cleve and Daniel A. Spielman. Exponential algorithmic speedup by a quantum walk. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 59–68, 2003.

[7] Konstantin Artiouchine, Philippe Baptiste, and Christoph Dürr. Runway scheduling with holding loop. In *Discrete Optimization Methods in Production and Logistics (DOM)*, pages 96–101, Omsk-Irkutsk, Russia, 2004.

[8] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.

[9] Abhinav Bahadur, Christoph Dürr, Ragav Kulkarni, and Thibault Lafaye. Quantum query complexity in computational geometry revisited. manuscrit non-publié, 2005.

[10] Philippe Baptiste. Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times. *Journal of Scheduling*, 2:245–252, 1999.

[11] Philippe Baptiste. Scheduling equal-length jobs on identical parallel machines. *Discrete Appl. Math.*, 1:21–32, 2000.

[12] Philippe Baptiste, Peter Brucker, Sigrid Knust, and Vadim Timkovsky. Ten notes on equal-execution-time scheduling. *4OR: Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, 2:111–127, 2004.

[13] Philippe Baptiste, Marek Chrobak, Christoph Dürr, Wojciech Jawor, and Nodari Vakhania. Preemptive scheduling of equal-length jobs to maximize weighted throughput. *Oper. Res. Lett.*, 32(3):258–264, 2004.

[14] Philippe Baptiste, Marek Chrobak, Christoph Dürr, and Francis Sourd. Preemptive multi-machine scheduling of equal-length jobs to minimize the average flow time. sousmis.

[15] Elena Barcucci, Alberto Del Lungo, Maurice Nivat, and Renzo Pinzani. Reconstructing convex polyominoes from horizontal and vertical projections. *Theoretical Computer Science*, 155(2):321–347, 1996.

[16] Elena Barcucci, Alberto Del Lungo, Maurice Nivat, and Renzo Pinzani. Medians of polyominoes: a property for the reconstruction. *International Journal of Imaging Systems and Technology*, 8:69–77, 1998.

[17] Howard Barnum, Michael E. Saks, and Mario Szegedy. Quantum decision trees and semidefinite programming. In *Proceedings of 18th IEEE Conference on Computational Complexity*, pages 179–193, 2003.

[18] Alexandre M. Bayen and Claire J. Tomlin. Real-time discrete control law synthesis for hybrid systems using milp: Application to congested airspace. In *Proceedings of the* American Control Conference, 2003.

[19] Robert Beals, Harry Buhrman, Richard Cleve, Michele Mosca, and Ronald de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48(4):778–797, 2001.

[20] Aija Berzina, Andrej Dubrovsky, Rusins Freivalds, Lelde Lace, and Oksana Scegulnaja. Quantum query complexity for some graph problems. In *Proceedings of the 30th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pages 140–150, 2004.

[21] O. Boruvka. O jistem problemu minimaln im. *Prace Mor. Prrodove Spol. v Brne (Acta Societ. Scient. Natur. Moravicae)*, 3:37–58, 1926.

[22] Michel Boyer, Gilles Brassard, Peter Hoyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte Der Physik*, 46(4-5):493–505, 1998.

[23] Gilles Brassard and Peter Hoyer. An exact quantum polynomial-time algorithm for Simon's problem. In *Proc. of Fifth Israeli Symposium on Theory of Computing and Systems (ISTCS)*, pages 12–23, 1997.

[24] Gilles Brassard, Peter Hoyer, Michel Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Quantum Computation and Quantum Information: A Millennuim Volume, AMS Contemporary Mathematical Series*, 305, 2002.

[25] Gilles Brassard, Peter Hoyer, and Alain Tapp. Quantum cryptanalysis of hash and claw-free functions. In LNCS 1380, editor, *Proc. of the Latin American Symposium on Theoretical Informatics (LATIN)*, pages 163–169, 1998.

[26] Richard A. Brualdi. Matrices of zeros and ones with fixed row and column sum vectors. *Linear Algebra and Applications*, 33:159–231, 1980.

[27] Peter Brucker and Sigrid Knust. Preemption can make parallel machine scheduling problems hard. *Osnabrücker Schriften zur Mathematik*, page 211, 1999.

[28] Peter Brucker and Svetlana A. Kravchenko. Complexity of mean flow time scheduling problems with release dates. Technical report, University of Osnabrück, 2004.

[29] Peter Brucker and Svetlana A. Kravchenko. Scheduling jobs with equal processing times and time windows on identical parallel machines. personal communication, 2005.

[30] Harry Buhrman, Christoph Dürr, Mark Heiligman, Peter Hoyer, Frédéric Magniez, Miklos Santha, and Ronald de Wolf. Quantum algorithms for element distinctness. In *IEEE Conference on Computational Complexity*, pages 131–137, 2001.

[31] Jacques Carlier. Problèmes d'ordonnancement à durées égales. *QUESTIO*, 5(4):219–228, 1981.

[32] Marek Chrobak, Peter Couperus, Christoph Dürr, and Gerhard Woeginger. On tiling under tomographic constraints. *Theoretical Computer Science*, 290(3):2125–2136, 2003.

[33] Marek Chrobak and Christoph Dürr. Reconstructing hv-convex polyominoes from orthogonal projections. *Information Processing Letters*, 69,:283–289, 1999.

[34] Marek Chrobak and Christoph Dürr. Reconstructing polyatomic structures from discrete X-rays: NP-completeness proof for three atoms. *Theoretical Computer Science*, 259(1-2):81–98, 2001.

[35] Marek Chrobak, Christoph Dürr, Wojciech Jawor, Lukasz Kowalik, and Maciej Kurowski. A note on scheduling equal-length jobs to maximize throughput. *Journal of Scheduling*, to appear.

[36] Ronald de Wolf. *Quantum Computing and Communication Complexity*. PhD thesis, Université de Amsterdam, 2001.

[37] Christoph Dürr. *Automates cellulaires quantiques*. PhD thesis, Université de Paris-Sud, 1997.

[38] Christoph Dürr, Eric Goles, Ivan Rapaport, and Eric Rémila. Tiling with bars under tomographic constraints. *Theoretical Computer Science*, 290(3):1317–1329, 2003.

[39] Christoph Dürr, Mark Heiligman, Peter Hoyer, and Mehdi Mhalla. Quantum query complexity of some graph problems. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 481–493, 2004.

[40] Christoph Dürr, Ivan Rapaport, and Guillaume Theyssier. Cellular automata and communication complexity. *tcs*, 322:355–368, 2004.

[41] Alex Fabrikant, Christos Papadimitriou, and Kunal Talwar. The complexity of pure nash equilibria. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*, 2004.

[42] Richard Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21:467–488, 1982.

[43] Richard Feynman. Quantum mechanical computers. *Foundations of Physics*, 16:507, 1986.

[44] Andrea Frosini and G. Simi. The NP-completeness of a tomographical problem on bicolored domino tilings. *Theoretical Computer Science*, 1–3:447–454, 2004.

[45] Anka Gajentaan and Mark H. Overmars. On a class of $o(n^2)$ problems in computational geometry. *Comput. Geom. Theory Appl.*, 5:165–185, 1995.

[46] Richard J. Gardner. *Geometric Tomography*. Cambridge University Press, 1995.

[47] Richard J. Gardner, Peter Gritzmann, and Dieter Prangenberg. On the computational complexity of determining polyatomic structures by X-rays. *Theoretical Computer Science*, 33(1-2):91–106, 2000.

[48] Michael R. Garey, David S. Johnson, Barbara B. Simons, and Robert Endre Tarjan. Scheduling unit-time tasks with arbitrary release times and deadlines. *SIAM Journal on Computing*, 10(2):256–269, 1981.

[49] Lov Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Annual ACM Symposium on Theory of Computing (STOC)*, pages 212–210, 1996.

[50] Gabor T Herman and Attila Kuba. *Discrete tomography : foundations, algorithms, and applications*. Birkhäuser, 1999.

[51] J.M. Keil. On the complexity of scheduling tasks with discrete starting times. *Operation Research Letters*, 12:293–295, 1992.

[52] Valerie King and M. Henzinger. Randomized dynamic graph algorithms with polylogarithmic time per operation. *Journal of the ACM*, 46(4):502–516, 1999.

[53] C. Kisielowski, P. Schwander, F.H. Baumann, M. Seibt, Y. Kim, and A. Ourmazd. An approach to quantitate high-resolution transmission electron microscopy of crystalline materials. *Ultramicroscopy*, 58:131–155, 1995.

[54] Pascal Koiran, Vincent Nesme, and Natacha Portier. A quantum lower bound for the query complexity of Simon's problem. Technical report, arxiv.org, 2005.

[55] D. Kölzow, A. Kuba, and A. Volčič. An algorithm for reconstructing convex bodies from their projections. *Discrete Comput. Geom.*, 4:205–237, 1989.

[56] Svetlana A. Kravchenko. On the complexity of minimizing the number of late jobs in unit time open shop. *Discrete Appl. Math.*, 100(1-2):127–132, 2000.

[57] Sophie Laplante, Troy Lee, and Mario Szegedy. The quantum adversary method and formula size lower bounds. In *IEEE Conference on Computational Complexity*, 2005.

[58] Sophie Laplante and Fredéric Magniez. Lower bounds for randomized and quantum query complexity using Kolmogorov arguments. In *Proceedings of 19th IEEE Conference on Computational Complexity*, pages 214–304, 2004.

[59] Eugene L. Lawler. Knapsack-like scheduling problems, the Moore-Hodgson algorithm and the 'tower of sets' property. *Mathl. Comput. Modelling*, 20(2):91–106, 1994.

[60] Joseph Y.-T. Leung and Gilbert H. Young. Preemptive scheduling to minimize mean weighted flow time. *Information Processing Letters*, 34:47–50, 1990.

[61] Fredéric Magniez, Miklos Santha, and Mario Szegedy. Quantum algorithms for the triangle problem. In *Proceedings of 16th ACM-SIAM Symposium on Discrete Algorithms*, 2005.

[62] Gatis Midrijanis. Exact quantum query complexity for total boolean functions. Technical report, arxiv.org, 2004.

[63] Ashwin Nayak. *Lower Bounds for Quantum Computation and Communication*. PhD thesis, University of California, Berkeley, 1999.

[64] Ashwin Nayak and Felix Wu. The quantum query complexity of approximating the median and related statistics. In *Proceedings of 31th Annual ACM Symposium on Theory of Computing (STOC)*, pages 384–393, 1999.

[65] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[66] Christophe Picouleau. Reconstruction of domino tiling from its two orthogonal projections. *Theoretical Computer Science*, 255:437–447, 2001.

[67] Herbert J. Ryser. *Combinatorial Mathematics*. Mathematical Association of America and Quinn & Boden, Rahway, New Jersey, 1963.

[68] Kunihiko Sadakane, Norito Sugawara, and Takeshi Tokuyama. Quantum algorithms for intersection and proximity problems. In *Algorithms and computation (Christchurch, 2001)*, volume 2223 of *Lecture Notes in Comput. Sci.*, pages 148–159. Springer, Berlin, 2001.

[69] Kunihiko Sadakane, Norito Sugawara, and Takeshi Tokuyama. Quantum computation in computational geometry. *Interdiscip. Inform. Sci.*, 8(2):129–136, 2002.

[70] P. Schwander, C. Kisielowski, M. Seigt, F.H. Baumann, Y. Kim, and A. Ourmazd. Mapping projected potential, interfacial roughness, and composition in general crystalline solids by quantitative transmission electron microscopy. *Physical Review Letters*, 71:4150–4153, 1993.

[71] Peter Shor. Polynomial-time algorithms for prime factorization and discrete logarithm on a quantum computer. *SIAM Journal on Computing*, 26:1484–1509, 1997.

[72] Barbara B. Simons. A fast algorithm for single processor scheduling. In *Proceedings IEEE 19th Annual Symposium on Foundations of Computer Science (FOCS'78)*, pages 246–252, 1978.

[73] Warren D. Smith. Sublinear time quantum algorithms in computational geometry and related areas. unpublished manuscript, 2001.

[74] Frits C.R. Spieksma and Yves Crama. The complexity of scheduling short tasks with few starting times. Technical report, Maastricht University, 1992. Reports in operation research and system theory.

[75] Gerhard J. Woeginger. The reconstruction of polyominoes from their orthogonal projections. *Information Processing Letters*, 77(5-6):225–229, March 2001. was previously a technical report at TU Graz, Austria, 1996.