Master Systèmes Dynamiques et Signaux

Mémoire de master

Conception de détecteurs quantiques optimaux via le calcul par intervalles

Auteur :
M. Pierre Engelstein

Encadrants :
Dr. Nicolas Delanoue
Pr. François Chapeau-Blondeau

Jury :
Pr. Laurent Hardouin
Dr. Nicolas Delanoue
Pr. François Chapeau-Blondeau
Pr. Sébastien Lahaye
Dr. Mehdi Lhommeau
Pr. David Rousseau

Version du
22 juin 2021

Remerciements

Je remercie Dr. Nicolas Delanoue et Pr. François Chapeau-Blondeau pour leur encadrement sur ce travail. Je remercie également mes parents pour les encouragements et l'aide apportés sur cette année de Master.

Table des matières

Table des figures

Chapitre 1

Introduction

Le problème de la détection optimale est un problème classique du traitement de l'information, issu des travaux sur les signaux dans la deuxième moitié du XX$^{\text{ème}}$ siècle. Le problème est de pouvoir détecter du mieux que l'on puisse un signal contenant de l'information parmi un signal plus ou moins quelconque : Alice envoie à Bob un message via un canal de transmition, et Bob doit pouvoir retrouver optimalement le message transmit par Alice.

On s'intéresse au problème équivalent dans le cadre de l'information quantique : Alice envoie à Bob un message codé sur un support quantique, et Bob doit pouvoir retrouver optimalement le message transmit. Cette fois-ci, on se place dans le cadre quantique, on est donc en présence d'un ensemble d'états quantiques possibles, et d'un ensemble d'opérateurs de mesures, formant le système à optimiser. L'objectif est, en connaissant les états d'entrée et leurs probabilités, de trouver l'ensemble des opérateurs de mesure qui forment une mesure optimale.

Ce problème peut se traiter en utilisant un certain nombre de critères de performance. Eldar, notamment, a utilisé le critère de la probabilité d'erreur de mesure en minimisation [1], ce qui revient à un problème de minimisation linéaire sous contraintes semi-définies positives, facile en pratique à résoudre. De même, Eldar a aussi considéré le critère de l'erreur quadratique de mesure [2], ce qui revient à un problème de minimisation quadratiques sous les mêmes contraintes semi-définies positives.

Nous nous intéressons ici au critère de l'information mutuelle [3], qui pose un problème de maximisation de fonction convexe sous contraintes semi-définies positives. Ce problème n'est en pratique pas facile à résoudre. Il peut l'être de façon approximative par exemple par des solutions de recuit. Nous nous intéressons à la résolution de ce problème de façon garantie, en utilisant le calcul par intervalle pour fournir un encadrement assuré du résultat optimal.

Ce rapport détaille dans un premier chapitre les éléments d'information

quantique relatifs au problème traité. On détaillera par la suite les notions de calcul par intervalle et l'algorithme d'optimisation utilisé. En dernier lieu, on détaillera le problème d'optimisation et l'application de l'analyse par intervalle à celui-ci.

Chapitre 2

Informatique quantique : éléments de base

Les notions de base d'informatique quantique sont décrites dans plusieurs ouvrages de référence, notamment dans [4, 5]. On présente ici un résumé des notions fondamentales à connaître pour la suite du rapport.

On pose 3 postulats, servant de base aux raisonnements qui suivront. Ces postulats sont confirmés jusqu'à présent par les expériences.

2.1 Etat d'un système quantique

Un système quantique peut être représenté par un vecteur d'état, de la même manière qu'un système physique classique. On le représente par la notation de Dirac, notée de la forme $|\psi\rangle$. Ce vecteur d'état est nécessairement de norme 1 (la somme des modules au carré vaut 1). On peut distinguer deux types d'états pour un système quantique : les états de base, formant une base orthonormée d'un espace vectoriel complexe, et les états superposés. Ces états superposés correspondent à une combinaison linéaire des états de base. On peut écrire généralement un état quantique de la façon suivante :

$$|\psi\rangle = \sum_i c_i \, |k_i\rangle \, , \tag{2.1}$$

avec les $|k_i\rangle$ états de base, et les c_i respectant $\sum_i |c_i|^2 = 1$ pour la normalisation du vecteur d'état.

L'état d'un système quantique peut être généralisé par une matrice densité, représentant un opérateur densité. Dans le cas d'un état pouvant être décrit par un vecteur d'état $|\psi\rangle$, état pur, l'opérateur densité ρ correspondant sera donné par $\rho = |\psi\rangle \langle\psi|$. Dans le cas d'un système complexe, on ne peut pas forcément factoriser l'opérateur ρ en fonction d'un $|\psi\rangle$ individuel, on parle alors d'état mixe.

Dans le cadre de l'informatique quantique, on utilise le système quantique le plus simple, appelé **qubit**. Ce système quantique est composé de deux états de base, $|0\rangle$ et $|1\rangle$, et des états superposés. Similairement à l'informatique classique, où on travaille sur le système physique le plus élémentaire - le bit - en quantique on travaille sur le système physique quantique élémentaire - le qubit. On dispose des mêmes états de base, mais l'informatique quantique apporte les états *intermédiaires* superposés. Dans la base canonique $\{|0\rangle, |1\rangle\}$, on note un qubit de la façon suivante : $|\psi\rangle = \alpha \cdot |0\rangle + \beta \cdot |1\rangle$.

2.2 Mesure quantique

Ces systèmes quantiques, physiques, doivent pouvoir être mesurés afin d'avoir une utilité. On distingue deux types de mesures : la mesure projective, et la mesure généralisée (aussi appelée POVM, *Positive Operator-Values Measure*). La mesure d'un système quantique passe par l'utilisation d'opérateurs de mesure, matrices densité au même titre que les états quantiques, donc se situant dans un espace de Hilbert de dimension N.

Mesure projective

La mesure projective est la forme la plus simple, constituée par un ensemble de N opérateurs de projection orthogonaux de rang 1 $\{\Pi\}$ avec $\Pi_n = |n\rangle \langle n|$. Ces opérateurs doivent vérifier la somme à l'identité : $\sum_n \Pi_n = I_n$.

Lors de cette mesure, la probabilité d'obtenir l'état $\Pi_k = |k\rangle \langle k|$ en mesurant un état ρ_j est donné par :

$$\Pr(|k\rangle) = \mathrm{tr}(\rho_j \Pi_k) \tag{2.2}$$

Cette mesure correspond à la mesure d'un état quantique sur les états de base $\{|0\rangle, |1\rangle\}$ lorsqu'il est représentable sous forme d'état quantique non mixe. La probabilité de mesurer $|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ en ayant $|\psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$ est alors donné par :

$$\Pr(|0\rangle) = \mathrm{tr}(\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} |\alpha|^2 & |\alpha||\beta| \\ |\alpha||\beta| & |\beta|^2 \end{bmatrix}) = \mathrm{tr}(\begin{bmatrix} |\alpha|^2 & 0 \\ 0 & 0 \end{bmatrix}) \tag{2.3}$$

$$\Pr(|0\rangle) = |\alpha|^2 \tag{2.4}$$

Mesure généralisée

La mesure généralisée consiste similairement à un ensemble de N opérateurs de mesure $\{\Pi\}$ se sommant à l'identité. En revanche, ces opérateurs de mesure sont étendus à des opérateurs semi-défini positifs. De la même manière que les POVM sont la généralisation des opérateurs de projection, ils permettent de mesurer la généralisation des états quantiques qui ne sont alors pas nécessairement descriptibles par des opérateurs densité.

De la même manière que les opérateurs de projection, la probabilité d'obtenir Π_k en mesurant l'état quantique ρ_j est donné par $\Pr(\Pi_k) = \mathrm{tr}(\rho_j \Pi_k)$.

Il faut noter que, lorsqu'on fait la mesure, on projette réellement le système quantique dans l'état de base. Concrètement, si on a un état superposé qu'on mesure, il se place dans l'état de base qu'on mesure, et toutes les mesures successives qu'on fera sur ce qubit donneront le même résultat. La mesure fait donc perdre l'état qu'on avait auparavant.

2.3 Dynamique du système

Entre l'étape de préparation d'un système quantique et sa mesure, on peut venir appliquer une série de transformations qui permettent de faire évoluer le système et donc effectuer des calculs. Les détails sont disponibles en annexe A.

Chapitre 3

Calcul par intervalle : éléments de base

Le calcul par intervalle est décrit au départ dans les travaux de Ramon Moore [6]. L'utilité de ce mode de calcul vient des problèmes que représente le stockage des nombres réels dans nos ordinateurs via la norme `IEEE 754`. En effet, on sait avec cette norme facilement représenter une certaine quantité, finie de nombres réels tels que 0.5, sous la forme $\texttt{signe} \times \texttt{base}^{\texttt{exposant}} \times (1 + \texttt{mantisse})$. Il est en revanche impossible de représenter précisément le reste des nombres réels, tels que 0.1. De ce fait, lorsqu'on se place dans des contextes de calculs, il devient évident qu'on peut se retrouver à accumuler des erreurs de précision qui vont venir fausser les résultats. Quand on veut garantir des résultats, par exemple sur des problèmes d'optimisation, cela peut devenir pénalisant.

3.1 Les intervalles

3.1.1 Intervalle et boite

Définition 1. *On définit un intervalle* $[\underline{x}, \overline{x}]$ *comme l'ensemble des nombres réels* x *tels que* $\underline{x} \leq x \leq \overline{x}$.

On note par la suite plus généralement $[x] = [\underline{x}, \overline{x}]$.

Exemple 1. *Si on veut représenter le nombre* $\sqrt{2} = 1.4142\ldots$, *on peut dire :* $1.4 \leq \sqrt{2} \leq 1.5$, *donc encadrer ce nombre par l'intervalle* $[1.4, 1.5]$.

On étend cette notion d'intervalle à plusieurs variables en prenant le produit cartésien de plusieurs intervalles pour former des boites en n dimensions :

Définition 2. *Une boite* $[\boldsymbol{x}]$ *est le produit cartésien des intervalles qui le composent :* $[\boldsymbol{x}] = ([x_1] \times [x_2] \times \cdots \times [x_n])$.

3.1.2 Fonctions d'inclusion

Avec cette notion d'intervalle, on peut définir le comportement quand on applique une fonction. L'idée est de se dire que, pour un intervalle ou une boite d'entrée $[x]$, l'intervalle image par une fonction f doit contenir l'ensemble des images prises par la fonction f pour tout les $x \in [x]$:

Définition 3. *Soit $f : \mathbb{R}^n \to \mathbb{R}^m$ une fonction, la fonction $[f] : \mathbb{R}^n \to \mathbb{R}^m$ est une **fonction d'inclusion** pour f si*

$$\forall [x] \in \mathbb{R}^n, f([x]) \subset [f]([x]) \tag{3.1}$$

Exemple 2. *La figure 3.1 montre l'encadrement d'une fonction $y = f(x)$ quelconque. La boite bleue est le produit cartésien de $[x]$ et de $[y]$.*

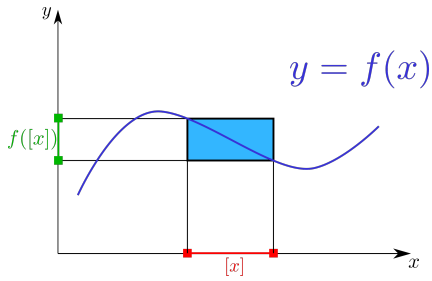


FIGURE 3.1 – Fonction d'inclusion

L'encadrement nécessite la connaissance précise de la forme de la fonction pour pouvoir l'encadrer correctement. Ceci peut se révéler compliquer pour des fonctions non-évidentes, typiquement quand on monte en dimension. Pour cela, on peut combiner les fonctions d'inclusion sans perdre la garantie d'inclusion, comme indiqué dans le théorème 1.

Théorème 1. *si $[f]$ et $[g]$ sont des fonctions d'inclusion respectives pour f et g, alors $[f] \circ [g]$ est une fonction d'inclusion pour $f \circ g$.*

Cela permet en pratique de construire des fonctions d'inclusions élémentaires puis de les combiner. En effectuant cette opération, on peut en revanche perdre de la précision sur l'encadrement comme le montre la figure 3.2.

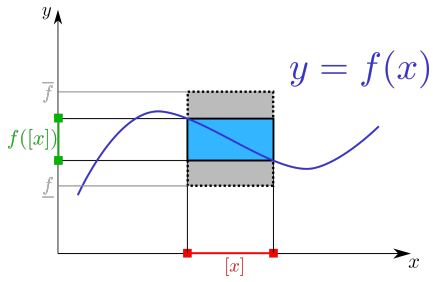


FIGURE 3.2 – Fonction d'inclusion composée de moindre qualité

3.1.3 Arithmétique élémentaire

Comme dit précédemment, on peut construire les fonctions d'inclusions des fonctions nécessaires pour n'importe quel problème en calcul par intervalle. Spécifiquement, il est utile de définir un certain nombre de fonctions de base permettant de former les blocs de construction pour la formation de fonctions plus complexes. On peut ainsi définir les opérateurs binaires (l'addition, la soustraction, la multiplication, ...) ainsi que les opérateurs unaires (l'exponentielle, la puissance, le sinus, ...).

Exemple 3. *Un certain nombre de fonctions arithmétiques élémentaires peuvent être formulées :*

- $[x_1] + [x_2] = [\underline{x_1} + \underline{x_2}, \overline{x_1} + \overline{x_2}]$
- $[x_1] - [x_2] = [\underline{x_1} - \overline{x_2}, \overline{x_1} - \underline{x_2}]$
- $[x_1] \times [x_2] = [min(\underline{x_1}\underline{x_2}, \underline{x_1}\overline{x_2}, \overline{x_1}\underline{x_2}, \overline{x_1}\overline{x_2}), max(\underline{x_1}\underline{x_2}, \underline{x_1}\overline{x_2}, \overline{x_1}\underline{x_2}, \overline{x_1}\overline{x_2})]$
- $[x]^2 = [0, max(\underline{x}^2, \overline{x}^2)]$
- $e^{[x]} = [e^{\underline{x}}, e^{\overline{x}}]$
- ...

On peut étendre ces définitions à l'ensemble des fonctions strictement monotones : il est évident de se dire que, si une fonction $f(x)$ est strictement croissante, alors $[f]([x]) = [f(\underline{x}), f(\overline{x})]$, et de même pour une fonction décroissante. On peut alors construire des fonctions plus complexes, comme $f : x \mapsto x^3$ en découpant la définition de la fonction par morceaux.

3.2 Optimisation avec les intervalles

On met en place un algorithme d'optimisation utilisant le calcul par intervalle pour obtenir un encadrement garanti de la solution à notre problème.

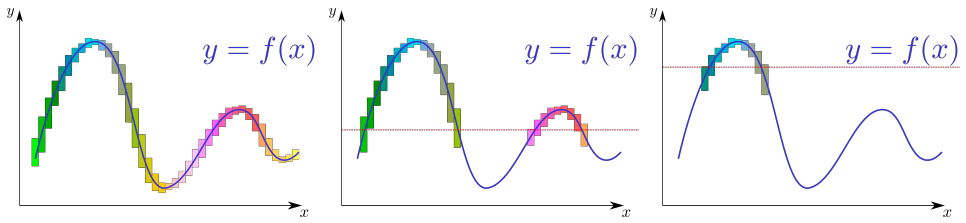
On veut résoudre le problème $\max(f(x))$ tel que $g(x) \leq 0$, avec f fonction coût et g ensemble des contraintes. Avec le calcul par intervalles, on cherche

à avoir un encadrement *garanti* de la solution au problème. Le principe de base est de découper l'ensemble des entrées en un certain nombre de boites, dépendant de la précision que l'on veut, comme à la figure 3.3a. On choisit ensuite un a solution admissible du problème suivant le théorème 2.

Théorème 2. *Soit un a une solution admissible du problème $\max_x f(x)$ tel que $g(x) \leq 0$ et x^* la solution optimale, on a :*

$$sup([f]([x])) \leq f(a) \Rightarrow x^* \notin [x] \tag{3.2}$$

Cela nous permet d'éliminer directement de l'ensemble des solutions les boites dont la borne supérieure de l'image est inférieure à l'image de ce critère a, puisque garanties comme ne contenant pas l'optimum du problème. La figure 3.3b illustre cette élimination. À l'issue de cette étape, on voit qu'on obtient un ensemble plus restreint de boites garanties comme contenant la solution, et on peut itérer en choisissant au fur et à mesure un critère a meilleur, et on arrive à un encadrement satisfaisant de la solution comme à la figure 3.3c.



(a) (b) (c)

FIGURE 3.3 – Optimisation naïve

Cette méthode d'optimisation, "naïve", permet d'obtenir un résultat satisfaisant, mais va être rapidement limitée si on veut des précisions plus élevées. En effet, on va avoir très rapidement un très grand nombre de boites à traiter. On peut remarquer entre autres qu'un certain nombre de boites vont être dans des "régions" globales pouvant être éliminées (par exemple, sur la figure 3.3b on a tout le bas de la fonction qui pourrait être éliminé d'un coup). Cela nous amène à un algorithme plus avancé, présenté en 3.4.

Au lieu de découper directement l'espace des entrées en un très grand nombre de boites, on va bisecter l'espace en deux boites, suivant l'axe le plus grand. On obtient deux boites, dont l'axe le plus grand aura été coupé en faisant $[x] \longrightarrow [\underline{x}; (\overline{x} + \underline{x}).0.5], [(\overline{x} + \underline{x}).0.5; \overline{x}]$. On considère la bissection par le milieu, mais on pourrait aussi utiliser des bissections plus avancées permettant d'accélérer l'algorithme.

Une fois obtenues ces deux boites, on peut évaluer la fonction d'inclusion et les contraintes, et décider de jeter les boites ne rentrant pas dans les contraintes. Sur l'ensemble des boites obtenues pour une itération, on effectue l'opération de recherche de critère décrit précédemment, et on réitère sur les boites restantes. Cela permet d'éliminer rapidement les boites de plus grande taille qui sont garanties ne comprenant pas l'optimum, et donc réduire considérablement le nombre de boites à traiter par la suite.

On considère l'algorithme fini quand on a obtenu une précision suffisante sur l'encadrement de la fonction ou des variables d'entrée, ou alors quand un certain nombre d'itérations ont été effectuées.

3.3 Implémentation

L'algorithme 3.4 a été implémenté en C# (dotnet 5) sur la base de la librairie IntSharp modifiée pour répondre à nos besoins (rajout de l'intervalle vide, de la fonction $x\log(x)$, des intervalles booléens, ...). Une interface graphique basique utilisant Blazor a été mise en place pour faciliter la visualisation de l'optimisation et des différents problèmes rencontrés lors du développement. L'ensemble du projet est disponible sur `https://github.com/PierreEngelstein/IntervalEval`. La solution est organisée en plusieurs modules :

- `IntervalEval` qui fourni la librairie de base pour le calcul par intervalle et l'optimisation ;
- `IntervalEval.Front` est l'interface web basique développée avec le framework Blazor Server ;
- `IntervalEval.Optimizer` est une interface en ligne de commande pour le problème spécifique détaillé dans ce rapport ;
- `IntervalEval.FrontConsole` est une interface en ligne de commande codée pour tester le problème à trois états quantiques d'entrée, pour tester les performances sur un problème en plus haute dimension ;
- `IntervalEval.Tests` fourni un ensemble de tests unitaires pour le bon fonctionnement de la librairie d'intervalle.

Data: $[I_{init}]$ initial search box ; ϵ stop criterion ; f cost function ; g constraints function ;
Output: $[f]$ bounds of best solution ; $[I]$ solution box
begin
  `solutions` list of solution boxes;
  Add $[I_{init}]$ to `solutions`;
  $[f_c]$ current bounds of solutions;
  while $\overline{f_c} - \underline{f_c} \geq \epsilon$ **do**
    `currentSolutions` empty list of boxes;
    `/* Bisect, evaluate cost, manage constraints */`
    forall *sol in solutions* **do**
      [left], [right] ⟵ bisect(sol);
      if *g*([*left*]) *is valid* **then**
        Add [left] to currentSolutions;
      end
      if *g*([*right*]) *is valid* **then**
        Add [right] to currentSolutions;
      end
    end
    `/* Remove boxes certified not to contain maximum */`
    Evaluate $[f]$ for all [currentSolutions];
    f_{best} best f(sol.mid) in all $[f]$;
    Remove all [sol] in [currentSolutions] where
     $sup([f]([\texttt{sol}])) \leq f_{best}$;
    solutions ⟵ currentSolutions
  end
  return *solutions*, $[f_c]$
end

FIGURE 3.4 – Algorithme de maximisation par le calcul par intervalle

Chapitre 4

Construction d'un détecteur optimal

On présente ici le détail du problème de la détection optimale quantique avec le critère de l'information mutuelle. Les résultats présentés ici ont fait l'œuvre d'une proposition de communication à la journée "Traitement du signal et applications quantiques" du GdR CNRS ISIS [7].

4.1 Formulation du problème

Le problème de la détection d'état quantique porte sur un ensemble de m états quantiques représentés par les opérateurs densité $\{\rho_i\ ,\ 1 \leq i \leq m\}$ munis des probabilités à priori $\{p_i \geq 0\ ,\ 1 \leq i \leq m\}$. L'objectif est d'obtenir un ensemble de m opérateurs de mesure $\{\Pi_j\ ,\ 1 \leq i \leq m\}$ permettant de mesurer le mieux possible par rapport aux probabilités les états d'entrée qui nous arrivent.

Les opérateurs ρ_i et Π_i sont des matrices Hermitiennes semi-définies positives, de la forme $\begin{bmatrix} a & b+ic \\ b-ic & d \end{bmatrix}$.

Plusieurs critères ont été proposés à optimiser afin de construire ces détecteurs optimaux. D'une part, on a la possibilité de travailler sur la minimisation de l'erreur quadratique de mesure [2] ou la maximisation de la probabilité de détection correcte [1]. D'autre part, et c'est ce sur quoi nous avons travaillé, on peut considérer le critère de l'information mutuelle comme critère à maximiser [3]. Ce critère indique la dépendance de deux variables aléatoires entre elles, il permet dans notre cas de bien caractériser la quantité d'information qu'on peut retirer des états d'entrée en ayant les opérateurs de mesure.

L'information mutuelle de deux variables aléatoires X et Y a été formulée par Shannon en 1948 [8]. Elle est donnée par :

$$I(X;Y) = \sum_{y\in Y}\sum_{x\in X} p_{(X,Y)}(x,y)\log\Big(\frac{p_{(X,Y)}(x,y)}{p_X(x)p_Y(y)}\Big), \tag{4.1}$$

mais peut aussi être écrite en fonction des entropies des variables aléatoires :

$$\begin{aligned} I(X;Y) &= H(X) - H(X|Y) && (4.2)\\ &= H(Y) - H(Y|X) && (4.3)\\ &= H(X) + H(Y) - H(X,Y). && (4.4) \end{aligned}$$

Avec $H(X)$ entropie marginale de X, $H(Y)$ entropie marginale de Y, $H(X|Y)$ entropie conditionelle de X sachant Y et enfin $H(X,Y)$ entropie jointe de X et Y. On peut utiliser indifférement $\log_2$, $\log_{10}$ ou ln pour le logarithme, le changement étant à une constante près. On utilise par la suite le logarithme base exponentielle pour l'ensemble des calculs.

Dans le cas classique, les entropies marginales, conditionnelles et jointes sont définies par :

$$\begin{aligned} H(X) &= -\sum_{x\in X} p(x)\log(p(x)), && (4.5)\\ H(Y) &= -\sum_{y\in Y} p(y)\log(p(y)), && (4.6)\\ H(X,Y) &= -\sum_{x\in X}\sum_{y\in Y} p(x,y)\log(p(x,y)), && (4.7)\\ H(Y|X) &= -\sum_{x\in X, y\in Y} p(x,y)\log\Big(\frac{p(x,y)}{p(x)}\Big) && (4.8) \end{aligned}$$

Dans le cas quantique, les formules restent les mêmes, mais on exprime les probabilités des variables en fonction des valeurs des états quantiques d'entrée.

En fonction d'un état d'entrée ρ_i de probabilité préalable p_i, et d'un opérateur de sortie Π_i, on peut définir leur probabilité jointe :

$$p(X = \rho_i, Y = \Pi_i) = p_i \operatorname{tr}(\rho_i \Pi_i). \tag{4.9}$$

On en déduit les probabilités marginales :

$$p(X = \rho_i) = \sum_j p_i \operatorname{tr}(\rho_i \Pi_j) \tag{4.10}$$

$$p(Y = \Pi_j) = \sum_i p_i \operatorname{tr}(\rho_i \Pi_j), \tag{4.11}$$

Et les probabilités conditionelles :

$$P(Y = \Pi_j | X = \rho_i) = \frac{\operatorname{tr}(\rho_i \Pi_j)}{\sum_k \operatorname{tr}(\rho_i \Pi_k)} \tag{4.12}$$

L'information mutuelle pour notre problème peut donc être ré-écrite de la façon suivante, en utilisant $\alpha_{ij} = \operatorname{tr}(\rho_i \Pi_j)$:

$$\begin{aligned} I(\rho; \Pi) &= H(\rho) + H(\Pi) - H(\rho, \Pi) \\ &= -\sum_{i=1}^{m} \Big(\sum_{j=1}^{m} \alpha_{ij} \Big) \log \Big(\sum_{j=1}^{m} \alpha_{ij} \Big) - \sum_{i=1}^{m} \Big(\sum_{j=1}^{m} \alpha_{ji} \Big) \log \Big(\sum_{j=1}^{m} \alpha_{ji} \Big) + \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_{ij} \log(\alpha_{ij}) \end{aligned} \tag{4.13}$$

On peut aussi exprimer l'information mutuelle en fonction de l'entropie conditionnelle, mais il est plus efficace d'utiliser celle donnée à l'équation 4.13 pour la résolution numérique.

Le problème se formule comme un problème de maximization de l'information mutuelle : on cherche à maximiser l'information qu'on peut obtenir sur ρ_i quand on a les opérateurs de mesure Π_i :

$$\max_{\Pi} I(\rho, \Pi) \tag{4.14}$$

tel que :

$$\Pi_j \succeq 0 \quad 1 \leq j \leq m \tag{4.15}$$

$$\sum_{j=1}^{m} \Pi_j = I \tag{4.16}$$

La contrainte 4.15 correspond à la semi-définition positive des opérateurs de mesure Π_j. Enfin, la contrainte 4.16 permet d'obtenir des opérateurs de

mesure cohérents pour que les probabilités de mesure $p(j) = \mathrm{tr}(\rho\Pi_j)$ soient positives et se somment à 1.

On est en présence d'une fonction convexe, et les contraintes engendrent un ensemble admissible convexe. C'est le cas idéal lors d'une minimisation, mais le problème est une maximisation, de même difficulté qu'une minimisation concave, on ne peut donc pas juste faire une descente de gradient pour le résoudre. On peut utiliser un certain nombre de méthodes approximatives, nous utilisons le calcul par intervalle afin d'obtenir un intervalle garantissant l'encadrement de la solution.

4.2 Convexité de l'information mutuelle

Davies considère dans [3] que l'information mutuelle pour ce problème peut être considérée comme étant convexe, simplifiant la résolution du problème en ayant à chercher le maximum sur les bords. On s'intéresse ici à l'étude de cette convexité.

Dans son article, Davies regroupe les traces et probabilités sous une seule variable $P_{ij} = p_i \,\mathrm{tr}(\rho_i\Pi_j)$. Ces coefficients P_{ij} forment une matrice des probabilités, telle que :

$$\sum_{ij} P_{ij} = 1, \tag{4.17}$$

$$\sum_{i} P_{ij} = p_i. \tag{4.18}$$

L'information mutuelle s'écrit donc :

$$I(P) = \sum_i H(\sum_j P_{ij}) + \sum_j H(\sum_i P_{ij}) - \sum_{ij} H(P_{ij}) \tag{4.19}$$

La fonction $H(x) = -x\log(x)$ est convexe, et donc I est convexe par rapport à la matrice des probabilités P. La figure 4.1 illustre cette fonction en fixant $p_1 = 0.3$ et $p_2 = 0.7$.

La convexité semble bien vraie par rapport à P, mais on cherche à optimiser les matrices Π_j. La matrice P comporte les traces de la multiplication $\rho_i\Pi_j$, qui est linéaire par rapport aux coefficients de Π_j. Si la fonction $I(P)$ est convexe par rapport à P, alors elle l'est par rapport aux Π_j, grace à la linéarité.

Quand on trace la même fonction, mais par rapport aux variables $\Pi_{k_{ij}}$, en se fixant dans un espace deux dimensions, on s'aperçoit que la fonction

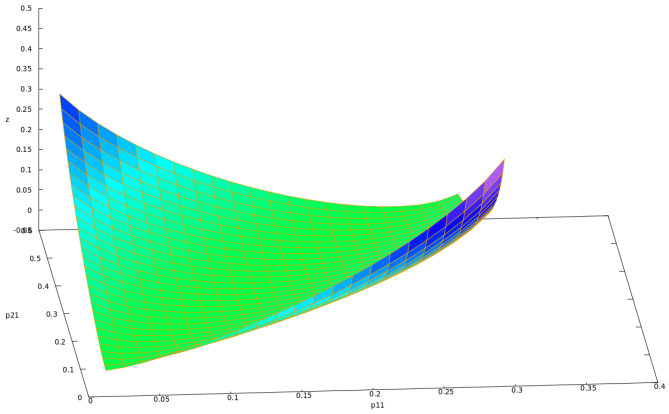


FIGURE 4.1 – Information mutuelle par rapport à la matrice de probabilités

n'est pas correctement définie sur les bords. Ceci est dû au fait que $x \log(x)$ n'est pas définit pour $x < 0$, ce qui fausse ou bloque les calculs, suivant l'implémentation. Le détail de notre implémentation est expliqué en **??**.

4.3 Formulation des contraintes

La définition du problème permet de résoudre notamment les cas immédiats des opérateurs de densité orthogonaux, mais la résolution devient très lente lorsqu'on passe à d'autres cas non orthogonaux. On rajoute des conditions au problème pour accélérer la résolution.

Le premier élément à simplifier est l'expression de l'entropie marginale de $X = \rho_i$. En effet, nous l'avons exprimé en fonction de la trace de la multiplication matricielle, mais on peut reprendre la définition donnée lors du cas classique qui dit que $H(X) = -\sum_{x \in X} p(x) \log(p(x))$. Le problème nous indique que nous connaissons les probabilités préalables des états d'entrée, on peut donc directement exprimer cette entropie en fonction de ces données et donc sans les variables de sortie Π_i.

Ensuite, on sait que les opérateurs de mesure se somment à l'identité. Cela signifie d'une part qu'on peut passer d'un problème à m matrices à un problème à $m - 1$ matrices pour $m \geq 2$. Les matrices étant carrées de dimension n, on passe de $m \times n^2$ variables à $(m - 1) \times n^2$ variables, ce qui est non négligeable.

De plus, le problème et les contraintes sont symétriques, on peut intervertir les Π_j sans influencer le résultat de la fonction coût. Cela nous permet

de couper le problème au moins en deux pour réduire à nouveau le temps de calcul. Du fait de la somme à l'identité, on peut ajouter en contrainte que $\Pi_{1_{1,1}} \leq \frac{1}{m}$ pour m opérateurs de mesure, puis $\Pi_{2_{1,1}} \leq \frac{1}{m-1}$, etc.

Enfin, pour rappel, les opérateurs de mesure sont des opérateurs densité, qui ne sont pas nécessairement purs. Pour qu'ils soient purs, il faudrait entre autres que $\mathrm{tr}(\Pi_i) = 1$. On peut considérer qu'on restreint le problème à un cas purs, et dans ce cas rajouter la contrainte que la somme des éléments diagonaux des opérateurs de mesure doit sommer à 1. Cela permet soit de retirer une variable par matrice densité au problème, en l'exprimant par $x_{n+1} = 1 - \sum_{i}^{n} x_i$ avec les x_i éléments diagonaux de l'opérateur densité, ce qui nécessite une reformulation du problème, soit l'ajout de la contrainte.

4.4 Exemple concret

4.4.1 Données du problème

Considérons deux états purs quantiques :

$$|\psi_1\rangle = \begin{bmatrix} \frac{1}{3} \\ \frac{2\sqrt{2}}{3} \end{bmatrix}, \quad |\psi_2\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}, \tag{4.20}$$

avec les probabilités préalables :

$$p(|\psi_1\rangle) = 0.1, \quad p(|\psi_2\rangle) = 0.9. \tag{4.21}$$

Ces deux états peuvent être réécris sous la forme d'opérateurs densité :

$$\rho_1 = \begin{bmatrix} \frac{1}{9} & \frac{2\sqrt{2}}{9} \\ \frac{2\sqrt{2}}{9} & \frac{8}{9} \end{bmatrix}, \quad \rho_2 = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} \tag{4.22}$$

On a aussi deux opérateurs de mesure inconnus en dimension 2, soit 8 variables inconnues :

$$\Pi_1 = \begin{bmatrix} a_1 & b_1 + ic_1 \\ b_1 - ic_1 & d_1 \end{bmatrix} \quad \Pi_2 \quad = \begin{bmatrix} a_2 & b_2 + ic_2 \\ b_2 - ic_2 & d_2 \end{bmatrix} \tag{4.23}$$

On peut déjà calculer le critère de Shannon maximum (*reference ?*) puisqu'on a la distribution de probabilités de X :

$$c_{max} = -\sum_j p(X_j)\log_2(p(X_j)) = -0.1\log_2(0.1) - 0.9\log_2(0.9) = 0.469, \tag{4.24}$$

qui signifie que quoi qu'il se passe, on ne pourra pas obtenir une information mutuelle supérieure à 0.469.

On veut résoudre le problème de la maximisation de l'information mutuelle en fonction des $\{\rho\}$ et des $\{\Pi\}$:

$$\begin{aligned}
&\max_{\Pi_1,\Pi_2} I(\rho_1,\rho_2,\Pi_1,\Pi_2) \qquad (4.25)\\
\Leftrightarrow \max_{\Pi_1,\Pi_2} \Big(&-(\alpha_{11}+\alpha_{12})\log(\alpha_{11}+\alpha_{12}) - (\alpha_{21}+\alpha_{22})\log(\alpha_{21}+\alpha_{22})\\
&-(\alpha_{11}+\alpha_{21})\log(\alpha_{11}+\alpha_{21}) - (\alpha_{12}+\alpha_{22})\log(\alpha_{12}+\alpha_{22})\\
&+(\alpha_{11})\log(\alpha_{11}) + (\alpha_{12})\log(\alpha_{12}) + (\alpha_{21})\log(\alpha_{21}) + (\alpha_{22})\log(\alpha_{22})\Big),\\
&\alpha_{jk} = \mathrm{tr}(\rho_j.\Pi_k)
\end{aligned}$$

tel que :

$$\Pi_1 \succeq 0, \Pi_2 \succeq 0 \tag{4.26}$$
$$\Pi_1 + \Pi_2 = I \tag{4.27}$$

On peut réduire considérablement le nombre de variables. En effet, l'équation 4.16 nous indique que les $\{\Pi\}$ se somment à l'identité, on peut donc réécrire Π_2 en fonction uniquement de Π_1 pour réduire à 4 variables $\{a_1, b_1, c_1, d_1\}$:

$$\Pi_2 = I_2 - \Pi_1 = \begin{bmatrix} 1-a_1 & -b_1 - ic_1 \\ -b_1 + ic_1 & 1-d_1 \end{bmatrix} \tag{4.28}$$

Enfin, on voit que les états quantiques d'entrée sont purs et ne comportent aucun terme complexe, ce qui permet immédiatement de retirer le terme c du système puisqu'il ne sera jamais pris en compte. On se retrouve donc au final avec 3 variables $\{a_1, b_1, d_1\}$.

On pose ensuite les contraintes sur ces variables. Tout d'abord, ces variables sont définies sur ces bornes spécifiques : $a_1 \in [0, 0.5]$, $b_1 \in [-1, 1]$, $d_1 \in [0, 1]$. La détermination de la semi-définition positive passe par les diagonales et le déterminant strictement positifs, d'une part avec les bornes précédentes et d'autre part avec deux nouvelles contraintes sur les 3 variables. Le problème s'écrit donc :

$$\max_{a_1,b_1,d_1} I(a_1, b_1, d_1),$$

tel que :

$$\begin{aligned} &a_1 \in [0, 0.5], b_1 \in [-1, 1], d_1 \in [0, 1], \\ &a_1 \times d_1 - b_1^2 \geq 0, \\ &(1 - a_1) \times (1 - d_1) - b_1^2 \geq 0. \end{aligned}$$

La résolution avec `ibex` ou avec notre solveur donne les deux opérateurs de mesure suivants :

$$\Pi_1 = \begin{bmatrix} 0.456 & -0.498 \\ -0.498 & 0.544 \end{bmatrix}, \quad \Pi_2 = \begin{bmatrix} 0.544 & 0.498 \\ 0.498 & 0.456 \end{bmatrix}, \tag{4.29}$$

Avec une information mutuelle $I(a_1, b_1, d_1) = 0.04723$. Notre optimiseur résout le problème en 6.4 secondes pour une précision sur I de 1×10^{-5}, et `ibexopt` résout en 88.3 secondes pour une précision sur I de 4×10^{-5}.

4.4.2 Résolution avec `ibex`

On peut tout d'abord résoudre le problème avec `ibexopt`. Les codes pour la résolution de ce problème sont disponibles en annexe C. On obtient un critère de Shannon de 0.156, avec une précision relative de 1×10^{-3}. On obtient les opérateurs de mesure suivant :

$\Pi_1 = \begin{bmatrix} 0.454 & -0.498 \\ -0.498 & 0.546 \end{bmatrix}, \quad \Pi_2 = \begin{bmatrix} 0.546 & 0.498 \\ 0.498 & 0.454 \end{bmatrix},$

avec un temps de calcul de 87 secondes.

Pour la résolution avec `ibex`, plusieurs modifications et ajouts doivent être faits. Tout d'abord, le langage utilisé par l'outil d'optimisation, `minibex`, ne considère que les problèmes de minimisation. Cela nécessite simplement de prendre la fonction opposée puisque $\max(f(x)) \Leftrightarrow \min(-f(x))$. On doit ensuite prendre l'opposé de l'intervalle solution pour obtenir le résultat final. Ensuite, le calcul par intervalle fait apparaître une difficulté quant à l'information mutuelle. On se retrouve à avoir de façon répétée des termes du type $x \times \log(x)$. Cette fonction est clairement définie sur $[0, +\infty]$ mais ne l'est pas sur $[-\infty, 0[$. De part le pessimisme du calcul par intervalle, on peut se retrouver à avoir des intervalles complètement négatifs ou contenant 0 en entrée de la fonction. On se retrouve donc avec un comportement mal défini dans certains cas. La solution pour ibex est de réimplémenter l'opérateur

`xlog` en considérant qu'en dehors ou sur le bord des bornes de définition on considère que la fonction renvoie 0.

Un autre point à noter est la grande variabilité des temps de calculs suivant les problèmes qui sont traités.

4.4.3 Résolution avec notre optimiseur

On a rencontré plusieurs problèmes avec ibex qui nous ont poussé à développer notre propre optimiseur.

Tout d'abord, en étudiant la fonction information mutuelle, on voit que cette fonction est convexe. Cela permet de considérablement réduire l'espace de définition en cherchant le maximum sur le bord de la fonction au lieu de chercher sur toute la fonction. Ceci n'est pas implémenté par défaut sur ibex.

Ensuite, sur l'implémentation de l'algorithme d'optimisation, on voit qu'`ibexopt` n'utilise qu'un seul thread pour son travail. On l'a vu en figure 3.4, la première boucle est facilement parallélisable, ce qui permet d'accélérer considérablement le temps de résolution, et ce en fonction du matériel à disposition.

Enfin, `ibexopt` ne permet pas l'accès aux boites en place au cours de l'évolution de l'algorithme. Cela ne permet pas une visualisation plus détaillée de ce qui est effectué, quelles régions sont enlevées au fur et à mesure. Dans le cas de problèmes "simples", ce n'est pas forcément important, en revanche avec un problème comme l'information mutuelle, qui possède beaucoup de redondance des variables d'entrée, c'est utile pour pouvoir avoir une idée des influences des contraintes. Un exemple de cette visualisation est disponible en figure 4.2. Les coordonnées $\{x, y, z\}$ correspondent aux variables $\{a_1, b_1, d_1\}$, et les boites sont colorées en fonction des contraintes satisfaites.

Notre optimiseur nous donne les mêmes solutions qu'ibex, à la différence qu'on arrive à une plus grande précision bien plus rapidement, en 13 secondes pour les mêmes données.

On peut noter entre autres, a la fois sur `ibexopt` et sur notre optimiseur que les temps de calculs dépendent du problème qu'on a. La figure 4.3 montre par exemple que, plus on s'approche de 90 degrés et des 180 degrés, qui sont des cas immédiats de problèmes orthogonaux, les temps de résolution sont bien moindres.

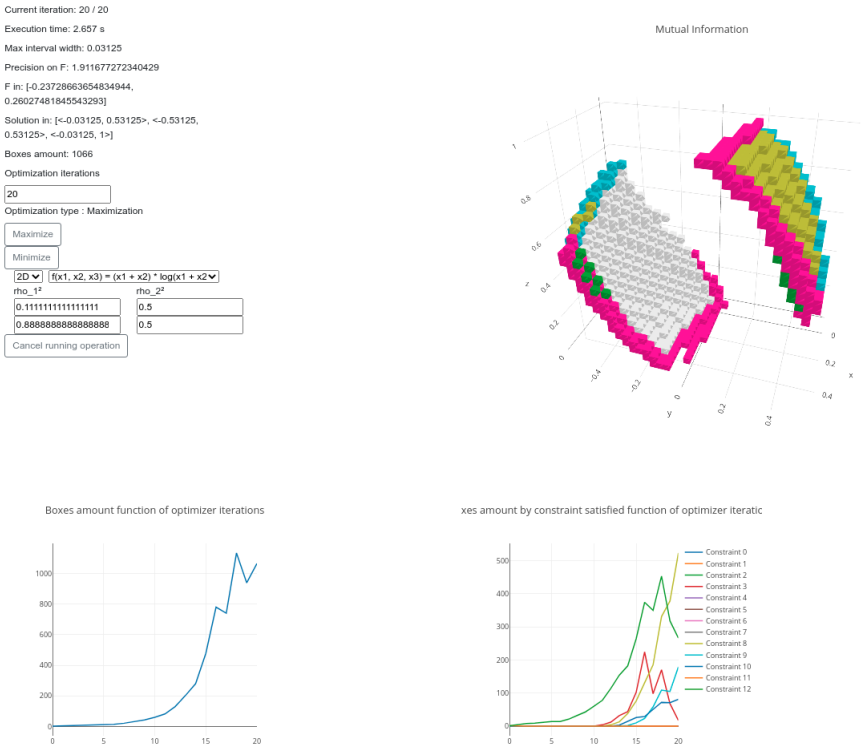


FIGURE 4.2 – Interface web de visualisation de l'optimisation

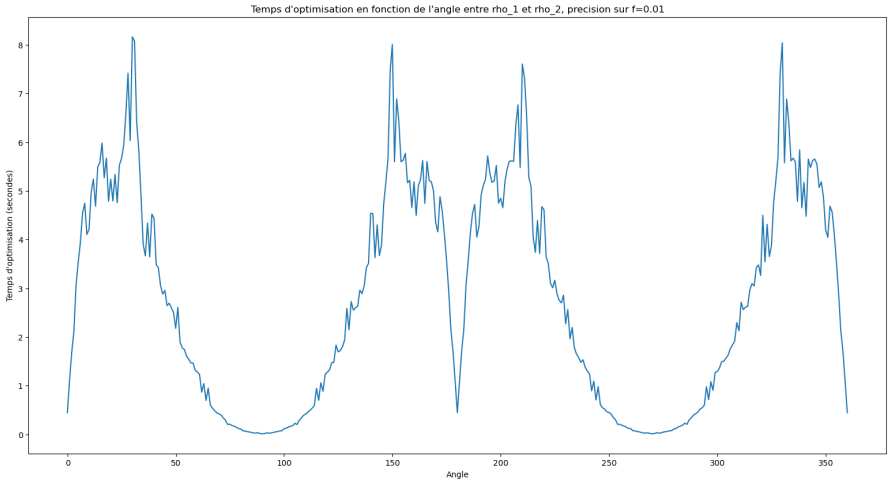


FIGURE 4.3 – Temps d'optimisation en fonction de l'angle entre les deux états quantiques

Chapitre 5

Conclusion

On a ainsi pu voir comment répondre au problème de la détection optimale quantique en utilisant un critère différent de ceux classiquement utilisés, l'information mutuelle. Une approche par intervalle, nouvelle pour cette question, a été appliquée pour obtenir un encadrement garanti de la solution.

Deux implémentations sont proposées, la première en utilisant un optimiseur existant, `ibexopt`, nécessitant certaines modifications pour s'adapter au problème posé. Un deuxième est proposé, en C#, améliorant entre autres la vitesse de résolution par rapport à `ibexopt`.

Ce problème de détection optimale n'a en revanche été traité que dans son cas le plus simple, en considérant un système "parfait" où aucun bruit n'apparaît. Il serait alors intéressant de voir plus loin en considérant les états d'entrée inconnus, bruités. Le problème serait alors de trouver la meilleure configuration états d'entrée - opérateurs de mesure pour que, en présence de bruit, on obtienne une communication optimale. Afin de résoudre ce problème, cependant, il est important de rappeler que l'algorithme présenté ne se dimensionne pas bien. Il serait alors aussi intéressant de coder la fonction $x \times \log(y)$ pour pouvoir utiliser la forme $H(X) - H(X|Y)$ de l'information mutuelle, qui pourrait présenter moins de redondance des variables d'entrées au travers des logarithmes et donc réduire le pessimisme de l'évaluation. Cette fonction est en 2 dimensions, donc moins évidente à représenter correctement par intervalles que la fonction simple $x \times \log(x)$.

Annexe A

Dynamique des systèmes quantiques

Comme n'importe quel système physique, on peut faire évoluer un système quantique dans le temps. Ici apparaissent deux propriétés. Tout d'abord, il découle du premier postulat 2.1 que la dynamique d'un système quantique doit conserver la norme unité. En effet, un état quantique doit, pour être valide, avoir une norme de 1, et donc l'évolution d'un système quantique d'un premier état vers un autre doit conserver cette unitarité. Cela veut dire que la matrice représentant l'évolution du système quantique doit respecter la propriété suivante :

$$U^\dagger U = UU^\dagger = I, \tag{A.1}$$

avec U la matrice d'évolution du système, $U^\dagger$ la matrice conjuguée transposée, ou adjointe, de U, et I l'identité.

Une deuxième propriété est également posée, ne découlant pas des deux postulats précédents. La dynamique d'un système quantique doit être aussi linéaire. Ainsi, on pourrait penser que n'importe quelle évolution unitaire serait valable, mais l'expérience nous montre que non, il faut en plus qu'elle soit linéaire.

En pratique, ces évolutions de systèmes quantiques peuvent être représentées par des portes logiques de façon similaire à la logique booléenne classique. Ces portes quantiques sont complètement caractérisées par la façon dont elles transforment les états quantiques dans la base canonique. On peut alors utiliser des tables de vérité pour les définir, de la même façon qu'en informatique classique :

1. La porte de Hadamard H. Elle permet de passer un qubit d'un état de base $|0\rangle$ à l'état superposé $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, ou de l'état de base $|1\rangle$ à l'état superposé $\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$. Elle est très utilisée en début de circuit pour préparer les qubits entrants dans un état permettant l'évaluation parallèle de toutes les entrées ;

2. Les portes de Pauli X, Y et Z permettant d'effectuer des rotations aux états des qubits ;
3. La porte de Toffoli, similaire d'un `NON` booléen à 3 qubit (il effectue un `NON` sur le dernier qubit quand les deux premiers sont à $|1\rangle$), est une porte universelle quantique [9]. Elle permet donc de construire l'ensemble des autres portes faisables.

Avec ces portes, on vient construire des circuits quantiques permettant de réaliser des algorithmes. L'annexe B montre un exemple de technique de réalisation de circuits. En algorithmes majeurs, on peut citer :

1. L'algorithme de Deutsch-Jozsa [10] qui permet de résoudre en une opération le problème de différentiation entre une fonction booléenne constante et une fonction booléenne équilibrée ;
2. L'algorithme de Grover [11] qui permet de résoudre en $\mathcal{O}(\sqrt{N})$ opérations le problème de recherche dans une liste non triée ;
3. L'algorithme de Shor [12] qui permet de résoudre le problème de factorisation en nombres premiers.

Annexe B

Création de circuits quantiques pour l'encodage de fonctions booléennes

On étudie ici la problématique de pouvoir construire systématiquement une fonction booléenne avec un ordinateur quantique, présenté par Younes et Miller en 2003 [13].

En informatique classique, l'ensemble des fonctions booléennes peuvent être décrites à l'aide des opérateurs **NAND** et **NOR**. Il s'agit donc de pouvoir les transcrire en quantique, et de pouvoir établir un système de combinaison de ces portes, pour permettre l'élaboration des circuits.

Dans ce modèle, on considère un registre de n qubits composant l'entrée du système, un registre de m qubits composant la sortie du système, et un registre de k qubits auxiliaires pour certaines opérations intermédiaires.

Pour cette construction, on se base sur la porte quantique **X** et ses équivalents composés **CNOT**, **CCNOT** (Toffoli), etc. On fourni alors un certain nombre de circuits de base pouvant être recomposés pour former des circuits plus complexes.

La compilation d'une fonction booléenne passe alors par 4 étapes :

1. Écriture de la table de vérité,
2. Pour chaque sortie donnant 1, former une porte **NOT** controlée. Chaque entrée va servir de contrôle, par 1 si l'entrée est à 1, et par 0 si l'entrée est à 0,
3. Développer le circuit résultant pour n'avoir que des portes **NOT** controlées par 0,
4. Simplifier le circuit en éliminant les doublons.

Étape 1 : établissement des premières portes contrôlées

La figure B.1 représente une porte **NOT** contrôlée. On note que les qubits de contrôle sont indiqués par • (contrôle par 1) et par ∘ (contrôle par 0). Le

dernier qubit est la cible (*target*). On effectue l'opération **NOT** sur la cible si et seulement si les bits de contrôles respectent leur condition (si il sont à 1 pour ceux qui sont contrôlés par 1, et si ils sont à 0 pour ceux contrôlés par 0).

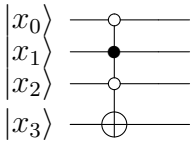


FIGURE B.1 – Porte **NOT** contrôlée

Cette porte en revanche ne peut pas être construite, on ne dispose en effet que des portes **NOT** contrôlées par 1 et pas de celles contrôlées par 0.

Exemple 4. *Soit la fonction booléenne* $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (x_3 \wedge \neg x_2) \vee (x_1 \wedge x_3)$. *Sa table de vérité est la suivante :*

x_1	x_2	x_3	$F(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

On a quatre sorties à 1. La figure B.2 représente donc le circuit initial qu'on obtient.

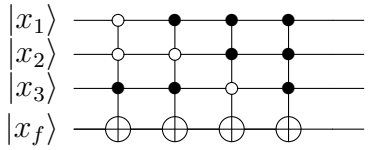


FIGURE B.2 – Circuit quantique pour $f(x_1, x_2, x_3)$

Étape 2 : Développement du circuit

La deuxième étape consiste à prendre le circuit obtenu précédemment et à le développer de façon à n'obtenir que des portes **NOT** contrôlées par 0. Dans le principe, une porte ayant un mélange de contrôle par 0 et par 1 va être équivalent à la combinaison des portes contrôlées par 0, qui vont avoir

des contrôles de moins en combinaison sur les contrôles par 1. Un exemple est plus clair pour comprendre. Reprenons la porte de la figure B.1. Elle est en fait équivalente au circuit B.3 :

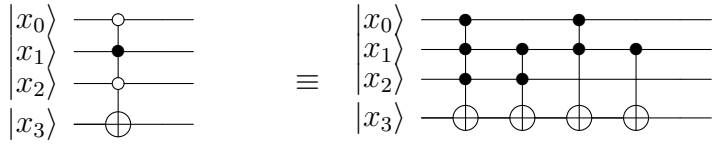


FIGURE B.3 – Équivalent sans contrôles par 0

Exemple 5. *On reprends notre exemple de fonction booléenne* $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee (x_3 \wedge \neg x_2) \vee (x_1 \wedge x_3)$*. Une fois développée, son circuit équivalent est illustré à la figure B.4*

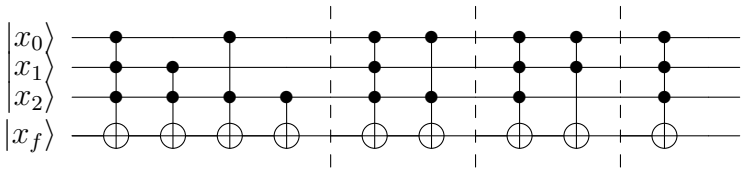
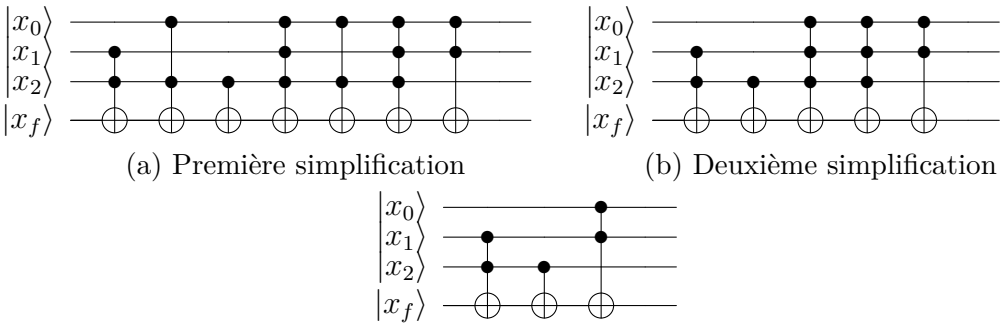


FIGURE B.4 – Circuit quantique développé pour $f(x_1, x_2, x_3)$

Étape 3 : Simplification du circuit

La dernière étape permet d'obtenir un circuit comportant le moins de portes possibles, en se basant sur le principe suivant : lorsqu'un circuit (ici composé de **CNOT**) est entouré de deux mêmes **CNOT**, alors celles-ci s'annulent et on peut alors enlever la paire doublon sans changer le résultat. En effectuant ce raisonnement récursivement, on arrive à obtenir un circuit minimal.

En reprenant l'exemple de la fonction booléenne précédente, on peut faire par étapes la simplification :



(a) Première simplification

(b) Deuxième simplification

(c) Dernière simplification

FIGURE B.5 – Simplifications successives pour $f(x_1, x_2, x_3)$

Extension à des circuits plus complexes

Avec cette méthode, il est facile de construire l'ensemble des fonctions booléennes à 2 bits, ainsi qu'a 3 bits. Pour construire des circuits plus complexes, on dispose alors de deux façons de procéder.

Tout d'abord, on peut refaire ces étapes sur la fonction booléenne plus complexe, et trouver un circuit minimal étant composé de n entrées, et d'un seul qubit auxiliaire pour la sortie.

En revanche, si on ne veut pas établir la table de vérité avant la construction du circuit, on peut utiliser les fonctions élémentaires et construire un circuit a partir de la chaine de caractère représentant la fonction. Il est important de noter que la méthode présentée ici ne modifie pas les entrées de la fonction, on peut donc les réutiliser comme on le souhaite pour venir greffer des fonctions supplémentaires au circuit. Cette méthode s'apparente algorithmiquement à la compilation des fonctions sur les ordinateurs classiques. En revanche, par rapport au circuit minimal qu'on pourrait trouver, on aura ici un qubit auxiliaire par sous-circuit, et un nombre de portes bien plus important. Pour des considérations de limitations matérielles qui sont pour le moment importantes, cette deuxième façon de procéder peut ne pas donner des résultats implémentables pour de trop grosses fonctions.

Annexe C

Codes développés pour `ibexopt`

C.1 Optimisation avec deux états d'entrée

```
Constants
    m_size = 2;
    P1[m_size][m_size] = ((0.0111111111, 0.0314269681); (0, 0.0888888889));
    P2[m_size][m_size] = ((0.45, 0.45); (0, 0.45));

// M: measurement operator
// P: density operator
function tr(M[m_size][m_size], P[m_size][m_size])
    return P(1)(1) * M(1)(1) + 2*M(1)(2)*P(1)(2) + 2*M(2)(1)*P(2)(1) + M(2)(2)*P(2)(2);
end

function EntropyM(_M1[m_size][m_size], _M2[m_size][m_size], _P1[m_size][m_size], _P2[
    m_size][m_size])
    sum_m1 = tr(_M1, _P1) + tr(_M1, _P2);
    sum_m2 = tr(_M2, _P1) + tr(_M2, _P2);
    return -xlog(sum_m1)-xlog(sum_m2);
end

function EntropyP(_M1[m_size][m_size], _M2[m_size][m_size], _P1[m_size][m_size], _P2[
    m_size][m_size])
    sum_p1 = tr(_M1, _P1) + tr(_M2, _P1);
    sum_p2 = tr(_M1, _P2) + tr(_M2, _P2);
    return -xlog(sum_p1)-xlog(sum_p2);
end

function EntropyMP(_M1[m_size][m_size], _M2[m_size][m_size], _P1[m_size][m_size], _P2[
    m_size][m_size])
    return -xlog(tr(_M1, _P1))-xlog(tr(_M1, _P2))-xlog(tr(_M2, _P1))-xlog(tr(_M2, _P2));
end

function MutualInformation(_M1[m_size][m_size], _M2[m_size][m_size], _P1[m_size][m_size
    ], _P2[m_size][m_size])
    return EntropyM(_M1, _M2, _P1, _P2) + ( -0.10 * ln(0.10) -0.90 * ln(0.90)) -
    EntropyMP(_M1, _M2, _P1, _P2);
end

function I(_P1[m_size][m_size], _P2[m_size][m_size], _m1a, _m1b, _m1c, _m1d)
    _M1 = ((_m1a, _m1b); (_m1c, _m1d));
    _M2 = ((1-_m1a, -_m1b); (-_m1c, 1-_m1d));
    return MutualInformation(_M1, _M2, _P1, _P2);
end

Variables
    M1_a in [0, 0.5], M1_b in [-1, 1], M1_d in [0, 1];

Minimize
    -I(P1, P2, M1_a, M1_b, 0, M1_d)

Constraints
    M1_a >= 0;
    M1_a <= 0.5;
    M1_d >= 0;
    M1_a*M1_d - M1_b^2 >= 0;
    (1-M1_a)*(1-M1_d) - (-M1_b)^2 >= 0;
end
```

C.2 Optimisation avec trois états d'entrée

```
Constants
    m_amount = 2;
    m_size = 2;
    P1[m_size][m_size] = ((0.10, 0); (0, 0));
    P2[m_size][m_size] = ((0.3, 0.3); (0, 0.3));
    P3[m_size][m_size] = ((0, 0); (0, 0.3));

// M: measurement operator
// P: density operator
function tr(M[m_size][m_size], P[m_size][m_size])
    return P(1)(1) * M(1)(1) + 2*M(1)(2)*P(1)(2) + 2*M(2)(1)*P(2)(1) + M(2)(2)*P(2)(2);
end

function EntropyM(_M1[m_size][m_size], _M2[m_size][m_size], _M3[m_size][m_size], _P1[
    m_size][m_size], _P2[m_size][m_size], _P3[m_size][m_size])
    sum_m1 = tr(_M1, _P1) + tr(_M1, _P2) + tr(_M1, _P3);
    sum_m2 = tr(_M2, _P1) + tr(_M2, _P2) + tr(_M2, _P3);
    sum_m3 = tr(_M3, _P1) + tr(_M3, _P2) + tr(_M3, _P3);
    return -xlog(sum_m1)-xlog(sum_m2) - xlog(sum_m3);
end

function EntropyP(_M1[m_size][m_size], _M2[m_size][m_size], _M3[m_size][m_size], _P1[
    m_size][m_size], _P2[m_size][m_size], _P3[m_size][m_size])
    sum_p1 = tr(_M1, _P1) + tr(_M2, _P1) + tr(_M3, _P1);
    sum_p2 = tr(_M1, _P2) + tr(_M2, _P2) + tr(_M3, _P2);
    sum_p3 = tr(_M1, _P3) + tr(_M2, _P3) + tr(_M3, _P3);
    return -xlog(sum_p1)-xlog(sum_p2)-xlog(sum_p3);
end

function EntropyMP(_M1[m_size][m_size], _M2[m_size][m_size], _M3[m_size][m_size], _P1[
    m_size][m_size], _P2[m_size][m_size], _P3[m_size][m_size])
    return -xlog(tr(_M1, _P1))-xlog(tr(_M1, _P2))-xlog(tr(_M1, _P3))-xlog(tr(_M2, _P1))-
    xlog(tr(_M2, _P2))-xlog(tr(_M2, _P3))-xlog(tr(_M3, _P1))-xlog(tr(_M3, _P2))-xlog(tr(
    _M3, _P3));
end

function MutualInformation(_M1[m_size][m_size], _M2[m_size][m_size], _M3[m_size][m_size
    ], _P1[m_size][m_size], _P2[m_size][m_size], _P3[m_size][m_size])
    return EntropyM(_M1, _M2, _M3, _P1, _P2, _P3) + ( -0.10 * ln(0.10) -0.60 * ln(0.60)
    -0.30 * ln(0.30)) - EntropyMP(_M1, _M2, _M3, _P1, _P2, _P3);
end

function I(_P1[m_size][m_size], _P2[m_size][m_size], _P3[m_size][m_size], _m1a, _m1b,
    _m1c, _m1d, _m2a, _m2b, _m2c, _m2d)
    _M1 = ((_m1a, _m1b); (_m1c, _m1d));
    _M2 = ((_m2a, _m2b); (_m2c, _m2d));
    _M3 = ((1-_m1a-_m2a, -_m1b-_m2b); (-_m1c-_m2c, 1-_m1d-_m2d));
    return MutualInformation(_M1, _M2, _M3, _P1, _P2, _P3);
end

Variables
    M1_a in [0, 1], M1_b in [-1, 1], M1_c in [-1, 1], M1_d in [0, 1];
    M2_a in [0, 1], M2_b in [-1, 1], M2_c in [-1, 1], M2_d in [0, 1];

Minimize
    -I(P1, P2, P3, M1_a, M1_b, M1_c, M1_d, M2_a, M2_b, M2_c, M2_d);

Constraints
    M1_a*M1_d - M1_b^2 - M1_c^2 >= 0;
    M1_a <= 1/3; // cost function & constraint both symmetric ((M1, M2) <=> (M2, M1))
    M2_a <= 0.5;
    M1_a + M1_d = 1;
    M2_a + M2_d = 1;
    M2_a*M2_d - M2_b^2 - M2_c^2 >= 0;
    (1-M1_a-M2_a) >= 0;
    (1-M1_a-M2_a) <= 1;
    (1-M1_d-M2_d) >= 0;
    (1-M1_d-M2_d) <= 1;
    (1-M1_a-M2_a)*(1-M1_d-M2_d) - (-M1_b-M2_b)^2 - (-M1_c-M2_c)^2 >= 0;
end
```

Bibliographie

[1] Y. C. Eldar, A. Megretski, and G. C. Verghese, "Designing optimal quantum detectors via semidefinite programming," *IEEE Transactions on Information Theory*, vol. 49, pp. 1007–1012, 2003.

[2] Y. C. Eldar and G. D. Forney, "On quantum detection and the square-root measurement," *IEEE Transactions on Information Theory*, vol. 47, pp. 858–872, 2001.

[3] E. B. Davies, "Information and quantum measurement," *IEEE Transactions on Information Theory*, vol. 24, pp. 596–599, 1978.

[4] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge : Cambridge University Press, 2000.

[5] D. N. Mermin, *Quantum Computer Science : An introduction*. Cambridge : Cambridge University Press, 2007.

[6] E. Moore, R, "Interval analysis," *Science*, vol. 158, no. 3799, pp. 365–365, 1967.

[7] P. Engelstein, N. Delanoue, and F. Chapeau-Blondeau, "Conception de détecteurs quantiques optimaux via le calcul par intervalles," Journée "Traitement du signal et applications quantiques" du GdR CNRS ISIS (Information Signal Image viSion). Prévue le 22 juin 2021 puis reportée.

[8] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[9] Y. Shi, "Both Toffoli and Controlled-NOT need little help to do universal quantum computation," 2002.

[10] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," *Proceedings of the Royal Society of London A*, vol. 439, pp. 553–558, 1992.

[11] L. K. Grover, *A Fast Quantum Mechanical Algorithm for Database Search*, p. 212–219. STOC '96, New York, NY, USA : Association for Computing Machinery, 1996.

[12] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, pp. 1484–1509, 1997.

[13] A. Younes and J. Miller, "Automated method for building cnot based quantum circuits for boolean functions," 05 2003.

Résumé — Ce mémoire s'intéresse à la question de la détection optimale quantique. On dispose d'un ensemble d'états quantiques que l'on veut détecter le plus optimalement possible par le biais d'un ensemble d'opérateurs de mesure. Plusieurs critères de performance existent tels que l'erreur de mesure ou l'erreur quadratique. On étudie ici le critère de l'information mutuelle, en y appliquant les techniques de calcul par intervalle pour garantir un encadrement du résultat.

Mots clés : Informatique quantique, détection optimale, opérateur de mesure, information mutuelle, calcul par intervalle.

Abstract — This report presents the problem of optimal quantum detection. We are provided with an set of quantum states with prior probabilities, and we want to build the set of quantum measurement operators to have an optimal measure. Multiple performance criterion exist, such as the probability of error of detection, or the square root error. We use the mutual information, using interval analysis to provide guaranteed bounds to the solution.

Keywords : Quantum computing, quantum optimal detection, measurement operator, mutual information, interval analysis.

Polytech Angers
62, avenue Notre Dame du Lac
49000 Angers