Comptage variationnel quantique

par

Julien Drapeau

Mémoire présenté au département de physique
en vue de l'obtention du grade de maître ès science (M.Sc.)

FACULTÉ des SCIENCES
UNIVERSITÉ de SHERBROOKE

Sherbrooke, Québec, Canada, 22 octobre 2024

Le _____

le jury a accepté le mémoire de M. Julien Drapeau dans sa version finale.

Membres du jury

Professeur Stefanos Kourtis
Directeur de recherche
Département de physique

Professeur André-Marie Tremblay
Membre interne
Département de physique

Professeur Baptiste Royer
Président rapporteur
Département de physique

À _____

Sommaire

Remerciements

Ma mère et mes soeurs,

Justin, Léanne, Thibault, Thomas, Philippe et William,

Sovannie,

Antoine, Benjamin, Jérémie, Martin, Pierre-Alexandre et tous les membres du groupe,

Stefanos,

Table des matières

Liste des figures

Introduction

Chapitre 1

Complexité du comptage

Il est souvent favorable de décrire la théorie de la complexité à l'aide du concept de *machine de Turing*. ... . Ce mémoire fera abstraction de ....

1.1 Classes de complexité

La théorie de la complexité s'intéresse à la classification des problèmes algorithmiques en *classes de complexité*, c'est-à-dire en ensembles de problèmes de même complexité. Classifier un problème permet de caractériser les limites atteignables de sa résolution par un algorithme.

Pourquoi est-ce qu'on veut faire ça ?

Comment est-il possible de généraliser la complexité d'un problème ? Pour ce faire, les classes de complexité se basent sur les ressources indispensables à la résolution du problème : le temps et la mémoire. Afin de trouver la solution à un problème, un programme doit effectuer un certain nombre d'opérations, limité dans le temps par le matériel informatique. On parle alors de *complexité en temps*. Afin de produire un résultat final, le programme garde souvent en mémoire les résultats intermédiaires. Tous les résultats doivent être sauvegardé dans le matériel informatique afin de pouvoir être réutilisés ultérieurement. La quantité d'information

conservée est aussi un facteur limitant pour le matériel informatique, on parle donc de *complexité en espace*.

Un problème de grande taille est plus *Parler de la dépendance en n.*

Le temps et la mémoire quantifie bien les ressources nécessaires des algorithmes. Par contre, ceux-ci dépendent du matériel informatique utilisé. Il est attendu qu'un ordinateur des temps modernes soit bien plus performant qu'une des premières machines analogues. Comment retirer cette dépendence dans la notion de complexité ? Pour ce faire, on fait appel à la notation asymptotique, communément appelé la notation $\mathcal{O}$.

Référence ici aux machines de Turing ?

Ces ressources permettent la séparation de plusieurs problèmes : il est en effet souhaitable d'être capable de séparer les algorithmes efficaces de ceux qui ne le sont pas.

Les relations entre les classes de complexité...

Les problèmes de décision

Qu'est-ce qui rend un algorithme efficace ? La classe de complexité P tente de répondre à cette question.

Comment décrire l'efficacité d'un algorithme ? Un algorithme est considéré comme efficace s'il résout un problème de la classe P.

Rajouter des exemples !

Définition 1.1 – Classe de complexité P

Une fonction A fait partie de la classe de complexité P si et seulement si un algorithme peut calculer $A(x) = y$ en temps polynomial, c'est-à-dire en temps $O(n^c)$ pour une taille $n = |x|$ et une constante c.

s

Définition 1.2 – Classe de complexité NP

Une fonction A fait partie de la classe de complexité NP si et seulement si une fonction $B \in \mathsf{P}$ existe tel que

$$A(x) = \exists y \mid B(x, y)$$

où $|y| = \text{poly}(|x|)$.

On appelle y le vérificateur (?).

Définition 1.3 – Classe de complexité #P

Un fonction A fait partie de la classe de complexité #P si et seulement si une fonction $B \in \mathsf{P}$ existe tel que

$$A(x) = |\{\, y \mid B(x, y) \,\}|$$

où $|y| = \text{poly}(|x|)$ pour toutes les valeurs y prises par $B(x, y)$.

Classes	Complexité	Description
...	...	...

Tableau 1.1 – Sommaire des classes de complexité.

Théoreme 1.1 – Théorème de Toda

...

1.2 Problème de satisfaisabilité booléenne

Plan

Le problème de *satisfaisabilité booléenne*, ou problème SAT est particulièrement important dans la théorie de la complexité. Montré comme NP-complet par le théorème de Cook-Levin [1], [2], il fut à la base de la définition de NP-complétude

et du problème $\mathsf{P} = \mathsf{NP}$. Celui-ci est aussi couramment utilisé dans la preuve de réductions de problèmes au sein de la classe de complexité NP.

Rajouter des sources et expliquer plus en détails le théorème de Cook-Levin.

Le problème SAT a une multitude d'applications, comme…, en partie grâce à la facilité de formuler ces applications à l'aide de formules propositionnelles.

Une formule propositionnelle, ou une expression booléene, est un ensemble de variables booléenes, $x_i \in \{0, 1\}$, reliées par des opérateurs booléeans de conjections ("ou", $\vee$), de disjonctions ("et", $\wedge$) ainsi que de négation ("non", $\neg$). Par exemple, l'expression $(x_1 \wedge x_2) \vee \neg x_3$ est un exemple de formule booléenne.

Dans l'étude du problème SAT, les formules propositionnelles sont souvent exprimées en forme normale conjonctive (CNF). Celle-ci consiste en

Définition 1.4 – Satisfaisabilité booléenne

Soit une constante $n \geq 1$ et une formule propositionnelle $\varphi(x_1, x_2, \ldots, x_n)$ où $x_i \in \{0, 1\}$. Existe-il une assignation des variables $x_1, x_2, \ldots, x_n$ telle que φ est satisfaisable, c'est-à-dire que $\varphi(x_1, x_2, \ldots, x_n) = 1$?

Exemple 1.1 – Satisfaisabilité booléenne

L

1.3 Intractabilité et approximations

Plan

1. Expliquer le concept d'intractabilité
2. Montrer la difficulté de résoudre des problèmes computationnels de manière exacte
3. Expliquer les advantages des méthodes approximatives (temps polynomial, applications réelles)
4. Introduire rigoureusement le concept d'approximation

Références

1.4 Complexité et bornes sur le comptage

Plan

1. Décrire les résultats actuels en terme de comptage exact et approximatif
2. Énumérer les algorithmes et les solveurs modernes (DPLL, *survey propagation*, *belief propagation*)
3. Mentionner les meilleures bornes sur les problèmes de comptage

Références

1. Wahlström, M. A Tighter Bound for Counting Max-Weight Solutions to 2SAT Instances. in Parameterized and Exact Computation (eds. Grohe, M. and Niedermeier, R.) 202–213 (Springer, Berlin, Heidelberg, 2008). doi :10.1007/978-3-540-79723-419.

2. Sinclair, A. and Jerrum, M. Approximate counting, uniform generation and rapidly mixing Markov chains. Information and Computation 82, 93–133 (1989).

1.5 Transitions de phase

Plan

1. Expliquer les différentes transitions de phase et leurs intuitions
2. Décrire l'objectif des algorithmes classiques locaux et globaux, comme le "belief propagation" ou le "survey propagation"
3. Expliquer brièvement où se situe VQCount par rapport à ça

Références

1. Watrous, J. Quantum Computational Complexity. Preprint at https ://doi.org/10.48550/arXiv.080 (2008).

2. Mézard, M. and Montanari, A. Information, Physics, and Computation. (Oxford University Press, Oxford, New York, 2009).

3. Survey propagation : An algorithm for satisfiability - Braunstein - 2005 - Random Structures amp; Algorithms - Wiley Online Library. https ://onlinelibrary.wiley.com/doi/abs/10.1002/rsa.20057.

Chapitre 2

Échantillonnage quasi aléatoire et comptage approximatif

Plan

1. Énoncer brièvement l'algorithme de JVV pour introduire la section
2. Re-mentionner l'importance du comptage approximatif (en autres en comparaison avec le comptage exact)
3. Mentionner les concepts nécessaires pour l'algorithme de JVV (auto-réductibilité, échantillonnage quasi aléatoire, comptage approximatif)

Références

2.1 Auto-réductibilité

L'auto-réductibilité (*self-reducibility*) est un concept complexe, essentiel à la compréhension du calcul et de la complexité, découlant de l'introduction de la réduction automatique (*autoreducibility*) **trakhtenbrotAutoreducibility1970**. Ces concepts sont particulièrement importants dans le contexte de génération aléatoire et du comptage approximatif, ainsi que pour la réduction entre le problème de décision et les problèmes de recherche.

La *réduction automatique* peut être introduite informellement de la manière suivante :

Définition 2.1 – Réduction automatique

Un problème algorithmique est dit *automatiquement réductible* s'il peut être résolu par un algorithme résolvant d'autres instances du même problème, sans que l'algorithme puisse interroger l'instance particulière qu'il cherche à résoudre.

Les problèmes automatiquement réductibles contiennent de l'information d'appartenance redondante, c'est-à-dire qu'il existe une structure dans l'ensemble de problèmes pouvant être exploitée pour simplifier le calcul d'une instance donnée. Ainsi, un algorithme peut résoudre une instance en utilisant l'information redondante présente dans d'autres instances, évitant ainsi les requêtes directes à l'instance en question. Connaître la solution à une autre problème peut alors aider la résolution du problème initial.

Exemple : Halting problem ?

Pour discuter de la génération aléatoire et le comptage approximatif, l'*auto-réductibilité descendante*, une forme limité de la réduction automatique, est une définition plus adéquate. En effet, cette condition est nécessaire à l'application de l'algorithme JVV. Informellement, on peut définir celle-ci comme :

Définition 2.2 – Auto-réductibilité descendante

Un problème algorithmique est dit *auto-réductible descendant* s'il peut être résolu grâce à un algorithme résolvant des instances de taille strictement inférieure.

Quelle est vraiment la différence entre auto-reducibility et self-reducibility ?

Cette propriété s'éclaircit en prenant le problème SAT comme exemple, qui s'avéra particulièrement utile dans la compréhension de l'algorithme JVV à la section 2.4. L'auto-réductibilité appliquée au problème de satisfaisabilité s'exprime facilement avec la relation suivante :

Relation 2.1 – Auto-réductibilité pour les problèmes SAT

Soit une constante $n \geq 1$ et une formule propositionelle $\varphi(x_1, x_2, \ldots, x_n)$ où $x_i \in \{ 0, 1 \}$. Alors,

$$\varphi(x_1, x_2, \ldots, x_n) = 1 \iff \varphi(x_1 = 0, x_2, \ldots, x_n) = 1 \vee \varphi(x_1 = 1, x_2, \ldots, x_n) = 1$$

Cette relation implique que l'ensemble de solutions d'une instance donnée peut être exprimée comme l'ensemble de solutions de deux instances plus petite du problème. *Élaborer...*

Plus précisément, le problème SAT est décrit comme une auto-réductibilité à longueur décroissante 2-disjonctive. Ici, 2-disjonctive fait référence à une formule propositionnelle dans une disjonction ($\vee$) d'une conjonction ($\wedge$) d'au plus deux variables. Longueur décroissante, ou descendante, signifie que l'algorithme résout des instances de taille strictement inférieure.

Une définition rigoureuse de l'auto-réductibilité peut aussi être pratique.

Définition 2.3 – Auto-réductibilité descendante

Soit Σ^* un ensemble fixé et fini encodant les instances d'un problème ainsi que leurs solutions. Soit $R \subseteq \Sigma^* \times \Sigma^*$ une relation binaire assignant à chaque instance de problème $x \in \Sigma^*$ un ensemble de solutions $R(x) = \{ y \in \Sigma^* \mid xRy \}$. Une relation $R \subseteq \Sigma^* \times \Sigma^*$ est auto-réductible si et seulement si

1. il existe une fonction calculable en temps polynomial $g \in \sigma^* \to \mathbb{N}$ tel que $xRy \implies |y| = g(x)$;
2. il existe une fonction calculable en temps polynomial $\psi \in \Sigma^* \times \Sigma^* \to \Sigma^*$ et $\sigma \in \Sigma^* \to \mathbb{N}$ satifaisant

$$\begin{aligned} &\sigma(x) = O(\log |x|) \\ &g(x) > 0 \Rightarrow \sigma(x) > 0 \quad \forall x \in \Sigma^\star \\ &|\psi(x, w)| \leqslant |x| \quad \forall x, w \in \Sigma^\star \end{aligned}$$

et tel que, pour tout $x \in \Sigma^*, y = y_1 \dots y_n \in \Sigma^*$,

$$\langle x, y_1 \dots y_n \rangle \in R \Leftrightarrow \left\langle \psi \left(x, y_1 \dots, y_{\sigma(x)} \right), y_{\sigma(x)+1} \dots y_n \right\rangle \in R$$

2.2 Échantillonnage quasi uniforme

Définition 2.4 – Échantillonneur quasi uniforme pleinement polynomial (FPAUS)

Un échantillonneur quasi uniforme pour un ensemble de solutions $S \subseteq \Sigma^* \times \Sigma^*$, où S représente la relation entre les instances d'un problème x et de ses solutions $y \in S(x)$, est un algorithme aléatoire prenant en entrée une instance $x \in \Sigma^*$ et une tolérance d'échantillonnage $\delta > 0$ et qui génère une solution $y \in S(x)$ tel que

$$\|Y - U|_{TV} \leq \delta$$

où Y est la distribution de probabilité de y et U est la distribution de probabilité uniforme sur $S(x)$. Si l'algorithme s'exécute en temps borné par un polynomial en $|x|$ et en $\ln(\delta^{-1})$, on parle d'échantillonneur quasi uniforme pleinement polynomial.

Changer la définition pour celle du papier de JVV ?

2.3 Comptage approximatif

Définition 2.5 – Schéma d'approximation aléatoire pleinement polynomial (FPRAS)

Un schéma d'approximation aléatoire pour un problème de comptage $f : \Sigma^* \to \mathbb{N}$ est un algorithme aléatoire prenant en entrée une instance d'un problème $x \in \Sigma^*$ et une tolérance d'erreur $\varepsilon > 0$ et qui génère un nombre $N \in \mathbb{N}$ tel que, pour toute instance x,

$$\Pr\left[(1+\varepsilon)^{-1} f(x) \leq N \leq (1+\varepsilon) f(x)\right] \geq \frac{3}{4}.$$

Si l'algorithme s'exécute en temps borné par un polynomial en $|x|$ et φ^{-1}, alors on parle de schéma d'approximation aléatoire pleinement polynomial.

2.4 Algorithme de Jerrum-Valiant-Vazirani

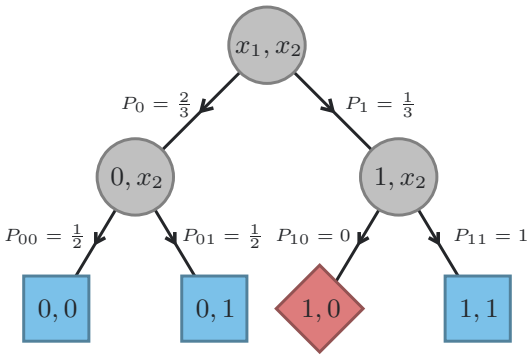

FIGURE 2.1

Chapitre 3

Algorithmes variationnels quantiques

3.1 Algorithme quantique d'optimisation approximative

Plan

1. Expliquer l'histoire et le lien avec le recuit quantique

Références

1. Farhi, E., Goldstone, J. and Gutmann, S. A Quantum Approximate Optimization Algorithm. Preprint at https ://doi.org/10.48550/arXiv.1411.4028 (2014).

2. Kadowaki, T. and Nishimori, H. Quantum annealing in the transverse Ising model. Phys. Rev. E 58, 5355–5363 (1998).

3. Finnila, A. B., Gomez, M. A., Sebenik, C., Stenson, C. and Doll, J. D. Quantum annealing : A new method for minimizing multidimensional functions. Chemical Physics Letters 219, 343–348 (1994).

4. Farhi, E. et al. A Quantum Adiabatic Evolution Algorithm Applied to Random Instances of an NP-Complete Problem. Science 292, 472–475 (2001).

5. Farhi, E., Goldstone, J., Gutmann, S. and Sipser, M. Quantum Computation by Adiabatic Evolution. Preprint at https ://doi.org/10.48550/arXiv.quant-ph/0001106 (2000).

3.1.1 Description de l'algorithme

Plan

1. Décrire le *Quantum Approximate Optimization Algorithm*

Références

1. Farhi, E., Goldstone, J. and Gutmann, S. A Quantum Approximate Optimization Algorithm. Preprint at https ://doi.org/10.48550/arXiv.1411.4028 (2014).

3.1.2 Initialisation des paramètres

Plan

1. Décrire l'importance d'une bonne initialisation des paramètres (*barren plateau*, non-convexité des paramètres)
2. Énumérer les principales méthodes
3. Expliquer l'initialisation aléatoire par grille
4. Expliquer *TQA*

Références

1. Bittel, L. and Kliesch, M. Training Variational Quantum Algorithms Is NP-Hard. Phys. Rev. Lett. 127, 120502 (2021).

2. Anschuetz, E. R. and Kiani, B. T. Beyond Barren Plateaus : Quantum Variational Algorithms Are Swamped With Traps. Nat Commun 13, 7760 (2022).

3. Akshay, V., Philathong, H., Morales, M. E. S. and Biamonte, J. D. Reachability Deficits in Quantum Approximate Optimization. Phys. Rev. Lett. 124, 090504 (2020).

4. Cain, M., Farhi, E., Gutmann, S., Ranard, D. and Tang, E. The QAOA gets stuck starting from a good classical string. Preprint at https ://doi.org/10.48550/arXiv.2207.05089 (2022).

Peut-être une référence de plus qui traite directement des barrens plateau ?

3.1.3 Encodage du problème

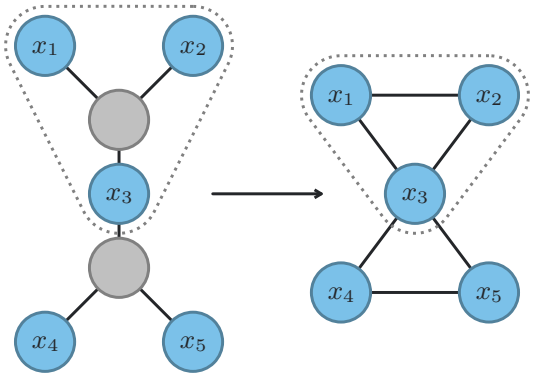

FIGURE 3.1

3.1.4 Choix du forçage

Plan

1. Expliquer le but du forçage
2. Expliquer forçage en X
3. Expliquer le forçage de Grover
4. Énumérer les forçages populaires

Références

3.2 Approche quantique des opérateurs alternants avec forçage de Grover

Plan

1. Décrire le *Quantum Alternating Operator Ansatz*
2. Décrire *Grover-Mixer Quantum Alternating Operator Ansatz*

Références

1. Hadfield, S. et al. From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz. Algorithms 12, 34 (2019).

2. Bärtschi, A. and Eidenbenz, S. Grover Mixers for QAOA : Shifting Complexity from Mixer Design to State Preparation. in 2020 IEEE International Conference on Quantum Computing and Engineering (QCE) 72–82 (2020). doi :10.1109/QCE49297.2020.00020.

3.3 Échantillonage et biais

Plan

1. Expliquer l'importance de l'échantillonnage non-biaisé
2. Expliquer le problème d'échantillonage associé au recuit quantique
3. Expliquer le problème d'échantillonage associé à QAOA
4. Expliquer pourquoi GM-QAOA résout ce problème (ne pas oublier d'expliquer les inconvénients de cette méthode)

Références

1. Zhang, Z. et al. Grover-QAOA for 3-SAT : Quadratic Speedup, Fair-Sampling, and Parameter Clustering. Preprint at https ://doi.org/10.48550/arXiv.2402.02585 (2024).

2. Mandrà, S., Zhu, Z. and Katzgraber, H. G. Exponentially Biased Ground-State Sampling of Quantum Annealing Machines with Transverse-Field Driving Hamiltonians. Phys. Rev. Lett. 118, 070502 (2017).

3. Matsuda, Y., Nishimori, H. and Katzgraber, H. G. Ground-state statistics from annealing algorithms : quantum versus classical approaches. New J. Phys. 11, 073021 (2009).

Plus de sources sur le fair sampling pour QAOA ?

Chapitre 4

Comptage variationnel quantique

Plan

1. Expliquer que cette section ne concerne que GM-QAOA

4.1 Auto-réductibilité des algorithmes variationnels quantiques

Plan

1. Faire la preuve de l'auto-réductibilité des algorithmes variationnels quantiques

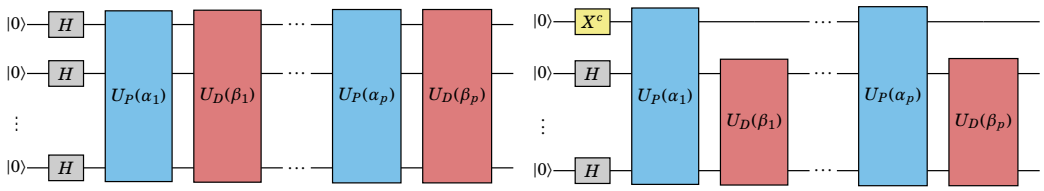

FIGURE 4.1

Références

4.2 Algorithme VQCount

Plan

1. Vulgariser l'algorithme de manière général
2. Expliquer rigoureusement l'algorithme
3. Faire le lien entre la notation utilisée dans les algorithmes de comptage classique
4. Faire la comparaison avec les travaux précédents
5. Rajouter l'algorithme complet en pseudo-code

Références

4.3 Module VQCount

Plan

1. Expliquer les librairies python dévelopées
2. Décrire *qaoa-quimb*
3. Décrire *VQCount*

Références

Chapitre 5

Résolution de problèmes #P-difficile

Plan

1. Mentionner les problèmes étudiés
2. Mentionner les méthodes utilisées (réseaux de tenseurs)
3. Décrire les paramètres de l'étude (nombre d'instances, régimes de complexité, etc.)
4. Expliquer les principaux résultats (compromis entre QAOA et GM-QAOA)

Références

5.1 Biais d'échantillonage des problèmes #P-difficile

Plan

1. Décrire le comportement de la non-uniformité pour les différents problèmes (ne pas oublier 2SAT)

Références

5.2 Performance et comportement de l'algorithme VQ-Count

Plan

1. Décrire le taux de réussite et le nombre d'échantillons requis
2. Décrire la performance de l'algorithme en fonction de la profondeur du circuit
3. Décrire l'efficacité d'échantillonage et la précision du compte
4. Présenter brièvement le ratio d'approximation

Références

5.3 Transitions de phase observées par l'algorithme VQCount

Plan

1.

Références

Conclusion

Annexe A

Réseaux de tenseurs

A.1 Simulation de circuits quantiques

Plan

1. Décrire le lien entre les circuits quantiques et les réseaux de tenseurs
2. Décrire les différentes méthodes de simulation (MPS-MPO et réseau de tenseurs général)
3. Décrire la méthode d'échantillonage

Références

1. Gray, J. quimb : A python package for quantum information and many-body calculations. Journal of Open Source Software 3, 819 (2018).

2. Ferris, A. J. and Vidal, G. Perfect Sampling with Unitary Tensor Networks. Phys. Rev. B 85, 165146 (2012).

Bibliographie

[1] S. A. Cook, « The Complexity of Theorem-Proving Procedures, » in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, sér. STOC '71, New York, NY, USA : Association for Computing Machinery, mai 1971, p. 151-158, ISBN : 978-1-4503-7464-4. DOI : 10.1145/800157.805047. (visité le 20/10/2024).

[2] L. A. Levin, « Universal Sequential Search Problems, »