Optimization of fault-tolerant quantum computing by the ZX-calculus

Vivien Vandaele

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Optimisation du calcul quantique tolérant aux fautes par le ZX-calculus

THÈSE

présentée et soutenue publiquement le 2 juin 2025

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Vivien Vandaele

Composition du jury

Président :	Pierrick Gaudry	CNRS, LORIA
Rapporteurs :	Dan Browne	University College London
	Peter Selinger	Dalhousie University
Examinateurs :	Matthew Amy	Simon Fraser University
	Stacey Jeffery	CWI, QuSoft, University of Amsterdam
	Elham Kashefi	CNRS, LIP6, University of Edinburgh
Invités :	Simon Martiel	IBM Quantum, IBM France Lab
	Maxime Remaud	Eviden Quantum Lab
Encadrants :	Simon Perdrix	Inria, LORIA
	Christophe Vuillot	Alice & Bob, Inria, LORIA

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503
Eviden Quantum Lab

Mis en page avec la classe thesul.

Remerciements

Je tiens tout d'abord à remercier l'ensemble de mes encadrants pour avoir mis en place un environnement propice au bon déroulement de cette thèse.

Merci à Simon Martiel de m'avoir encadré dès mes premiers pas dans l'informatique quantique, et pour avoir veillé à ce que ce parcours continue en toute sérénité.

Merci à Cyril Allouche pour la confiance qu'il m'a accordée tout au long de mon parcours au sein d'Eviden, et pour avoir assuré avec bienveillance la continuité de mon encadrement côté industriel.

Merci à Simon Perdrix pour ses conseils avisés, l'apport de son expertise, ainsi que sa rigueur scientifique.

Merci à Christophe Vuillot pour son implication constante et pour le partage de ses précieuses connaissances.

Je tiens à remercier Dan Browne et Peter Selinger pour avoir accepté d'être rapporteurs. Je suis également reconnaissant envers les autres membres de mon jury : Matthew Amy, Pierrick Gaudry, Stacey Jeffery et Elham Kashefi.

Mes remerciements vont également à tous les membres de l'équipe R&D d'Eviden et de l'équipe MOCQUA que j'ai eu le plaisir de côtoyer durant ces années. Je remercie notamment Maxime Remaud pour notre collaboration fructueuse en fin de thèse.

Enfin, merci à mes parents pour leur soutien indéfectible.

Introduction en Français

Qu'il soit classique ou quantique, un ordinateur repose sur deux composants fondamentaux : le matériel et le logiciel. Ces éléments forment une relation symbiotique, chacun dépendant de l'autre pour que le système fonctionne efficacement. Le matériel fournit l'infrastructure physique, comprenant les processeurs, les unités de mémoire, les dispositifs de stockage et les interfaces d'entrée-sortie. Il établit les capacités et les limitations fondamentales du système. L'architecture du matériel détermine les opérations de base qui peuvent être effectuées et la vitesse à laquelle elles peuvent être exécutées. Le logiciel, quant à lui, est la composante intangible qui exploite le potentiel du matériel. Il se compose de programmes, de données et d'instructions qui dirigent les opérations du matériel. L'aspect logiciel s'étend du code machine de bas niveau, qui interagit directement avec le matériel, aux langages de programmation de haut niveau qui abstraient les complexités des opérations matérielles.

Au cœur de l'interaction matériel-logiciel se trouve le processus de compilation. La compilation sert de pont crucial entre les abstractions de haut niveau du logiciel et les opérations de bas niveau exécutables sur le matériel. C'est le processus par lequel le code source lisible par l'homme est transformé en instructions exécutables par la machine. Dans l'informatique classique, les compilateurs ont été affinés pendant des décennies pour optimiser le code pour diverses architectures, en équilibrant des facteurs tels que la vitesse d'exécution, l'utilisation de la mémoire et la consommation d'énergie. En revanche, la compilation pour l'informatique quantique en est encore à ses débuts. Bien que l'objectif fondamental de traduire le code de haut niveau en instructions machines reste le même, l'informatique quantique présente des défis et des opportunités uniques qui nécessitent une refonte radicale des stratégies de compilation et le développement de nouvelles méthodes. Nous sommes, en substance, chargés de construire une pile de compilation entièrement nouvelle et innovante dans son intégralité.

La création d'une pile de compilation pour l'informatique quantique implique notamment le développement de nouveaux langages de programmation, la conception de représentations intermédiaires spécifiques au quantique et la création de nouvelles techniques d'optimisation qui tiennent compte des particularités des systèmes quantiques. Un autre aspect critique de la compilation quantique est l'intégration de techniques de correction d'erreurs et de protocoles tolérants aux fautes. Ceux-ci sont essentiels pour contrer les effets de la décohérence et du bruit quantique, rendant ainsi le calcul quantique plus fiable et robuste. De plus, les compilateurs quantiques doivent être adaptables aux architectures matérielles quantiques en rapide évolution et capables d'exploiter les avantages uniques offerts par différentes technologies matérielles. Relever ces défis est crucial pour combler le fossé entre les algorithmes quantiques théoriques et leur exécution réelle sur les infrastructure matérielles quantiques. Ainsi, la compilation joue un rôle central dans l'évaluation du coût du calcul quantique tolérant aux fautes et l'amélioration de sa faisabilité et de sa praticité.

Cette thèse aborde certains de ces défis de compilation dans le contexte de l'informatique quantique à grande échelle. Nous présentons des algorithmes efficients, des procédures automa-

tisées et des méthodes formelles pour optimiser le calcul quantique tolérant aux fautes. Pour y parvenir, nous utilisons une large variété de modèles pour représenter le calcul quantique, allant des formalismes abstraits aux représentations de bas niveau. Parmi ces modèles, il y a entre autres des représentations de haut niveau de certains opérateurs quantiques, le modèle de circuits quantiques et le ZX-calcul, un langage graphique rigoureux conçu pour raisonner sur les processus quantiques. L'utilisation de ces diverses représentations nous permet de concevoir des stratégies d'optimisation qui ciblent différents niveaux d'abstraction dans le processus de compilation, contribuant ainsi au développement d'une pile de compilation complète et performante pour l'informatique quantique tolérant aux fautes.

Les résultats de cette thèse sont présentés selon l'ordre dans lequel ils interviendraient dans cette chaîne de compilation, suivant le flux naturel allant des opérations de haut niveau jusqu'aux instructions exécutables par le matériel. La première étape considérée est la synthèse de circuits quantiques. Un circuit quantique est un modèle de calcul quantique qui représente les opérations quantiques comme une séquence d'opérations élémentaires appelées portes quantiques et mesures. Ces portes quantiques sont les éléments constitutifs du calcul quantique, analogues aux portes logiques classiques mais opérant sur des bits quantiques (qubits). Alors que certaines portes classiques, comme AND et OR, effectuent des opérations irréversibles sur les bits classiques, les portes quantiques sont des opérations unitaires qui transforment de manière réversible les états quantiques. Elles peuvent créer et manipuler des phénomènes quantiques tels que la superposition et l'intrication, qui n'ont pas d'équivalents classiques. Le processus de synthèse de circuits quantiques consiste à traduire des opérations quantiques abstraites, souvent exprimées sous forme mathématique, en séquences de portes quantiques élémentaires. La phase de synthèse établit l'implémentation de base qui servira d'entrée aux procédures d'optimisation ultérieures.

Le choix des portes quantiques disponibles pour la synthèse, appelé jeu de portes ou ensemble de portes, impacte significativement à la fois le processus de synthèse et l'implémentation du circuit qui en résulte. Les ensembles de portes élémentaires courants comprennent les rotations mono-qubit et les opérations contrôlées à deux qubits. Un ensemble de portes est dit universel si toute opération unitaire peut être exactement implémentée à l'aide d'une séquence finie de portes de cet ensemble. Cependant, il s'avère qu'aucun ensemble fini de portes donnant lieu à un ensemble fini d'opérations unitaires ne peut être universel. Cela conduit à la notion d'universalité approximative : un ensemble de portes est approximativement universel si toute opération unitaire peut être approximée avec une précision arbitraire en utilisant une séquence finie de portes de cet ensemble. Cette notion est suffisante pour les besoins pratiques, car nous pouvons toujours choisir une précision d'approximation rendant toute différence expérimentale indétectable.

Le théorème de Solovay-Kitaev, un résultat fondamental en informatique quantique, garantit que toute porte quantique peut être approximée avec une précision arbitraire en utilisant un ensemble fini de portes approximativement universel. De plus, le théorème fournit un algorithme explicite pour trouver de telles approximations avec une complexité polylogarithmique dans le nombre de portes requises. Ce théorème garantit que nous pouvons, en principe, implémenter n'importe quel algorithme quantique en utilisant uniquement un petit ensemble de portes élémentaires. Néanmoins, le choix de cet ensemble de portes approximativement universel a des implications pratiques profondes, particulièrement dans le contexte de la correction d'erreurs quantiques et du calcul quantique tolérant aux fautes.

Les systèmes quantiques sont intrinsèquement fragiles et hautement sensibles au bruit environnemental et aux imperfections opérationnelles. Pour faire face à ce défi fondamental, des codes correcteurs d'erreurs quantiques ont été développés pour protéger l'information quantique en l'encodant à travers plusieurs qubits physiques, formant ce que nous appelons des qubits

logiques. Bien que la correction d'erreur soit essentielle, elle n'est pas suffisante en soi. Nous devons également effectuer des opérations quantiques fiables sur ces qubits logiques tout en maintenant les propriétés de correction d'erreur du code, une caractéristique connue sous le nom de tolérance aux fautes.

Dans les codes correcteurs d'erreurs quantiques, seul un ensemble restreint de portes peut être implémenté de manière tolérante aux fautes. Certaines portes, appelées portes transversales, peuvent être implémentées directement sur les qubits logiques encodés tout en préservant les propriétés de correction d'erreur du code. Les opérations Clifford, générées par la porte CNOT (controlled-NOT) à deux qubits et les portes Hadamard (H) et de phase (S) à un qubit, admettent souvent des implémentations tolérantes aux fautes efficaces, par exemple à travers des opérations transversales. Cependant, les portes Clifford seules sont insuffisantes pour le calcul quantique universel. Pour atteindre l'universalité, les portes Clifford doivent être complétées par au moins une porte non-Clifford. Un choix courant est la porte T à un qubit, conduisant à l'ensemble de portes Clifford+T largement utilisé. L'implémentation tolérante aux fautes des portes non-Clifford telles que la porte T nécessite généralement des protocoles plus complexes impliquant la préparation d'états, la mesure et le retour d'information classique, les rendant plus coûteuses en ressources que les portes Clifford.

Cette contrainte fondamentale sur l'ensemble des portes utilisées pour atteindre la tolérance aux fautes, combinée à la disparité des coûts d'implémentation entre les différents types de portes, implique que le problème de synthèse ne peut pas être évalué uniquement sur le nombre total de portes. Au contraire, nous devons considérer à la fois l'implémentabilité tolérante aux fautes des portes dans notre code correcteur d'erreurs cible et leurs coûts d'implémentation respectifs.

Au-delà du nombre de portes, plusieurs métriques supplémentaires peuvent guider le processus de synthèse. La profondeur du circuit, correspondant à la longueur du plus long chemin à travers le circuit, impacte directement le temps d'exécution. Le nombre de qubits requis représente une contrainte fondamentale qui détermine les ressources physiques nécessaires pour exécuter le circuit sur le matériel quantique. Le coût espace-temps global combine ces facteurs pour estimer les besoins totaux en ressources. Les algorithmes de synthèse doivent donc équilibrer plusieurs objectifs d'optimisation, parfois concurrents, tout en travaillant avec un ensemble de portes restreint.

Une fois la synthèse des différentes parties d'un algorithme quantique effectuée, nous obtenons une implémentation sous forme de circuit quantique. La phase suivante dans la pile de compilation quantique considérée dans cette thèse est l'optimisation des circuits quantiques. L'optimisation de circuit prend la sortie de la phase de synthèse et la transforme en un circuit équivalent qui minimise des métriques de coût spécifiques tout en préservant sa fonctionnalité.

L'optimisation des circuits quantiques peut être abordée sous plusieurs angles. Une méthode courante consiste à s'appuyer sur des égalités de circuits quantiques et la reconnaissance de structure pour appliquer une série de transformations locales dans le circuit. Cette approche identifie des séquences spécifiques de portes qui peuvent être remplacées par des séquences équivalentes plus efficaces. Par exemple, les paires de portes CNOT agissant sur les mêmes qubits s'annulent mutuellement, et certaines séquences de portes de rotation à un qubit peuvent être combinées ou simplifiées. Bien que les règles individuelles de réécriture soient généralement simples, leur application répétée ou leur combinaison crée un vaste espace de transformations possibles.

Étant donné la croissance exponentielle de l'espace des transformations, une recherche exhaustive est généralement irréalisable, même pour des circuits quantiques de taille modérée. Cela motive le développement de stratégies efficaces pour naviguer efficacement dans l'espace des transformations. Une approche consiste à analyser le circuit pour concevoir des règles de réécriture personnalisées spécifiques à sa structure. Plutôt que de s'appuyer uniquement sur un

ensemble fixe d'identités, cette méthode crée des transformations qui sont adaptées au circuit ou à la catégorie de circuits en cours d'optimisation. Par exemple, cette approche a été utilisée dans des algorithmes efficaces pour réduire le nombre de portes T dans les circuits Clifford+T. En se concentrant sur les transformations susceptibles d'être pertinentes pour le circuit quantique donné, ces méthodes peuvent réduire considérablement l'espace de recherche et obtenir de meilleurs résultats d'optimisation dans un temps de calcul raisonnable.

Une approche alternative pour l'optimisation des circuits quantiques consiste à effectuer une resynthèse de certaines parties du circuit quantique. Plutôt que d'appliquer des règles de réécriture locales, la resynthèse extrait d'abord une description de plus haut niveau de l'opération effectuée par une section du circuit. Cette opération extraite est ensuite synthétisée à nouveau en utilisant des méthodes spécialisées qui ciblent les métriques d'optimisation souhaitées. Un avantage clé de la resynthèse réside dans son potentiel à dépasser les optima locaux en restructurant complètement le circuit. Comme les descriptions de haut niveau des parties extraites sont généralement invariantes sous les transformations locales du circuit, cette méthode d'optimisation est moins contrainte par la structure initiale du circuit.

Le choix de l'approche d'optimisation dépend de multiples facteurs, incluant les objectifs spécifiques de l'optimisation, les caractéristiques du circuit d'entrée telles que sa structure ou sa taille, et les moyens de calcul disponibles. De plus, l'interaction entre les différentes métriques d'optimisation ajoute un niveau supplémentaire de complexité au processus d'optimisation. Notamment, l'ordre dans lequel les différentes métriques d'optimisation sont traitées peut impacter significativement les résultats finaux. Par exemple, dans les circuits Clifford+T, l'optimisation du nombre de portes Hadamard avant la minimisation du nombre de portes T s'est avérée conduire à de meilleurs résultats globaux, à la fois en termes de nombre de portes T et de besoins en qubits auxiliaires. Ainsi, l'étude des interdépendances entre les différentes métriques est essentielle pour concevoir une stratégie d'optimisation efficace.

L'optimisation des circuits quantiques peut réduire significativement les besoins en ressources, mais le circuit optimisé doit ensuite être traduit en une implémentation tolérante aux fautes, ce qui présente des défis supplémentaires. Le modèle de circuits quantiques, bien que précieux pour la conception et l'analyse d'algorithmes, présente des limitations inhérentes lors de la représentation d'opérations tolérantes aux fautes au niveau logique. En particulier, son approche porte par porte et ses propriétés unitaires ne parviennent pas à capturer précisément la nature basée sur la mesure des protocoles tolérants aux fautes courants. Ces limitations motivent le développement et l'adoption de représentations alternatives qui s'étendent au-delà du modèle traditionnel de circuits quantiques.

Le ZX-calcul s'est révélé être une représentation particulièrement utile pour répondre à ces défis. Basé sur les réseaux de tenseurs, ce langage graphique représente les opérations quantiques sous forme de diagrammes avec des règles de réécriture bien définies qui préservent l'équivalence sémantique. De manière similaire aux identités de circuits qui permettent des transformations de circuits, ces règles de réécriture permettent une manipulation rigoureuse des diagrammes ZX. Cependant, le ZX-calcul offre une plus grande flexibilité grâce à ses blocs de construction fondamentaux qui peuvent représenter des opérations plus élémentaires que les portes quantiques. Notamment, ces blocs de construction du ZX-calcul ont une correspondance naturelle avec certaines opérations tolérantes aux fautes. En particulier, il a été démontré que le ZX-calcul est un langage pour la chirurgie de codes, une approche basée sur la mesure pour effectuer des opérations tolérantes aux fautes entre états quantiques encodés. Là où le modèle de circuit nécessiterait des séquences de portes plus complexes pour représenter de telles opérations, le ZX-calcul peut les capturer de manière plus directe et intuitive.

Le ZX-calcul peut également être utilisé pour établir des relations entre différents modèles de

calcul et problèmes d'optimisation. Par exemple, comme nous le démontrons dans cette thèse, l'optimisation du nombre de qubits à travers divers modèles de calcul quantique peut être efficacement reformulée en problèmes de théorie des graphes en s'appuyant sur le ZX-calcul. Une telle connexion donne accès à des techniques algorithmiques établies et des résultats théoriques issus de la théorie des graphes, offrant de nouvelles perspectives sur certains problèmes d'optimisation.

De plus, le ZX-calcul peut être utilisé comme représentation intermédiaire pour l'optimisation des circuits quantiques. Un circuit quantique peut être traduit en un ZX-diagramme, optimisé en utilisant les règles de réécriture du ZX-calcul, puis retraduit en circuit. Ce processus de transformation peut révéler des opportunités d'optimisation qui ne sont pas directement apparentes dans le modèle standard de circuit quantique.

En définitive, comme souligné dans cette introduction, l'approche et le modèle de représentation choisis dépendent de multiples facteurs, tels que le code correcteur d'erreurs quantiques sélectionné, le modèle de calcul utilisé pour réaliser le calcul quantique tolérant aux fautes, les métriques d'optimisation, les ressources de calcul disponibles et les contraintes de temps, ainsi que le niveau d'abstraction. Dans cette thèse, nous tirons parti des forces de ces diverses approches et représentations pour développer de nouveaux résultats pour la compilation quantique, avec un accent particulier sur l'optimisation des ressources pour le calcul quantique tolérant aux fautes.

Contributions et organisation du manuscrit. Ce manuscrit est structuré en quatre parties. La Partie I établit les fondements théoriques nécessaires à la compréhension des contributions présentées dans ce manuscrit :

- Le Chapitre 1 fournit une introduction à l'informatique quantique et au modèle des circuits quantiques, présentant les concepts clés et les notations utilisés tout au long de ce manuscrit.

- Le Chapitre 2 présente les principes fondamentaux de la correction d'erreurs quantiques et de l'informatique quantique tolérant aux fautes, en mettant l'accent sur la chirurgie de codes et le code de surface.

- Le Chapitre 3 introduit le ZX-calcul, un langage graphique utilisé dans cette thèse pour formuler et résoudre des problèmes d'optimisation.

Les contributions de cette thèse sont présentées dans les parties suivantes. La Figure 0 illustre comment les résultats présentés dans chaque chapitre contribuent aux différentes étapes de la pile de compilation quantique. La Partie II se concentre sur la synthèse de circuits quantiques pour les opérateurs arithmétiques, qui sont des éléments fondamentaux pour de nombreux algorithmes quantiques :

- Le Chapitre 4 introduit un nouveau circuit pour l'addition quantique, atteignant une profondeur polylogarithmique sans qubits auxiliaires.

- Le Chapitre 5 présente un multiplicateur quantique efficace pour les corps binaires, avec un nombre sous-quadratique de portes T et un faible coût espace-temps. Cette opération joue un rôle crucial dans la cryptanalyse quantique de la cryptographie des courbes elliptiques binaires.

La Partie III se concentre sur l'optimisation des circuits quantiques :

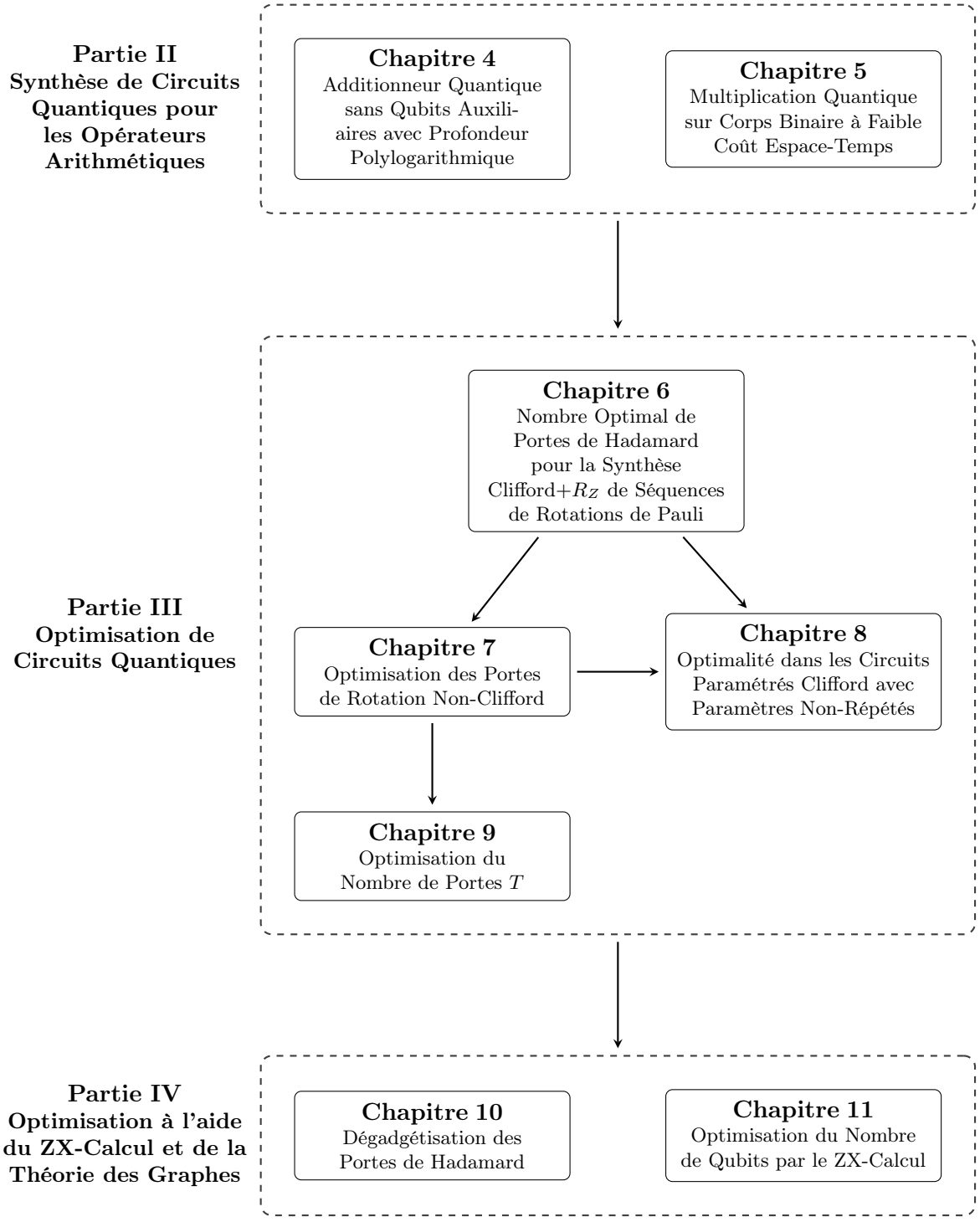


Figure 0: Pile de compilation formée par les résultats présentés dans cette thèse.

- Le Chapitre 6 présente un algorithme pour la synthèse de séquences de rotations de Pauli sur l'ensemble de portes Clifford+R_Z avec un nombre minimal de portes de Hadamard. Cette optimisation est particulièrement utile comme étape de prétraitement pour optimiser le nombre de portes T dans les circuits quantiques.

- Le Chapitre 7 introduit un algorithme rapide et efficace pour optimiser le nombre de portes de rotation non-Clifford sans nécessiter la connaissance des angles de rotation. Cette approche utilise notamment l'algorithme d'optimisation des portes de Hadamard présenté au Chapitre 6.

- Le Chapitre 8 établit l'optimalité des algorithmes présentés aux Chapitres 6 et 7 pour les circuits paramétrés Clifford avec des paramètres non répétés. Les circuits quantiques paramétrés sont des circuits quantiques qui incluent des paramètres ajustables dans leurs portes quantiques. De tels circuits apparaissent naturellement dans les algorithmes quantiques variationnels.

- Le Chapitre 9 présente des algorithmes pour optimiser le nombre de portes T dans les circuits quantiques. Nous démontrons que nos algorithmes produisent des circuits avec un nombre de T plus faible sur la plupart des circuits quantiques évalués tout en étant beaucoup plus rapides que les méthodes précédentes. Nous prouvons également une nouvelle borne supérieure constructive sur le nombre de portes T dans les circuits Clifford+T.

La Partie IV présente des techniques d'optimisation pour l'informatique quantique tolérant aux fautes en utilisant le ZX-calcul et la théorie des graphes :

- Le Chapitre 10 introduit une nouvelle méthode pour optimiser le nombre de qubits dans les circuits quantiques basée sur la dégadgétisation des portes Hadamard. Nous prouvons que ce problème d'optimisation est équivalent au problème bien connu du coupe-cycle de sommets en théorie des graphes.

- Le Chapitre 11 présente une approche pour optimiser le nombre de qubits dans divers modèles de calcul en s'appuyant sur le ZX-calcul. Nous montrons comment le problème d'optimisation du nombre de qubits est lié à des problèmes bien connus de la théorie des graphes, tels que le problème de décomposition en chemins de largeur minimale.

Les résultats présentés dans chaque chapitre sont basés sur les articles suivants :

- Chapter 4: [1] Maxime Remaud, and Vivien Vandaele. Ancilla-free Quantum Adder with Sublinear Depth. *arXiv preprint*, 2025. arXiv:2501.16802.

- Chapter 5: [2] Vivien Vandaele. Quantum binary field multiplication with subquadratic Toffoli gate count and low space-time cost. *arXiv preprint*, 2025. arXiv:2501.16136.

- Chapter 6: [3] Vivien Vandaele, Simon Martiel, Simon Perdrix, and Christophe Vuillot. Optimal Hadamard Gate Count for Clifford+T Synthesis of Pauli Rotations Sequences. *ACM Transactions on Quantum Computing*, 5(1):1–29, February 2024. arXiv:2302.07040. doi:10.1145/3639062.

- Chapters 7 and 8: [4] Vivien Vandaele, Simon Perdrix, and Christophe Vuillot. Optimal number of parametrized rotations and Hadamard gates in parametrized Clifford circuits with non-repeated parameters. *arXiv preprint*, 2024. arXiv:2407.07846.

- Chapter 9: [5] Vivien Vandaele. Lower T-count with faster algorithms. *arXiv preprint*, 2024. arXiv:2407.08695.
- Chapters 10 and 11: [6] Vivien Vandaele. Qubit-count optimization using ZX-calculus. *arXiv preprint*, 2024. arXiv:2407.10171.

Contents

Introduction en Français iii

List of Figures xvii

List of Tables xix

List of Algorithms xxi

Introduction 1

I Background 9

1 Quantum Computing and the Quantum Circuit Model 11

1.1 Introduction to quantum computing . . . 12

1.1.1 Mathematical framework and notations . . . 12

1.1.1.1 Dirac notation . . . 12

1.1.1.2 The computational, diagonal, and circular bases . . . 12

1.1.1.3 Combining ket vectors . . . 13

1.1.2 Quantum states . . . 13

1.1.2.1 Qubit . . . 13

1.1.2.2 Multi-qubit quantum states . . . 15

1.1.3 Quantum operators and measurements . . . 15

1.1.3.1 Unitary operators . . . 15

1.1.3.2 Quantum measurements . . . 16

1.2 Quantum gates and quantum circuits . . . 18

1.2.1 Quantum gates . . . 18

1.2.1.1 Pauli gates . . . 18

1.2.1.2 Clifford gates . . . 19

1.2.1.3 Universal gate sets . . . 21

1.2.2 Quantum circuits . . . 22

1.2.2.1 Quantum circuit diagrams . . . 22
1.2.2.2 Circuit identities . . . 23
1.2.2.3 Circuit metrics . . . 24
1.2.3 Global gates . . . 25
1.2.3.1 Multi-controlled NOT (MCX) gate . . . 25
1.2.3.2 Fan-out gate . . . 26
1.2.3.3 Ladder gate . . . 26
1.2.3.4 Pauli rotations . . . 28

2 Quantum Error Correction and Fault-Tolerant Quantum Computing 29
2.1 Quantum error correction . . . 30
2.1.1 Quantum error correction principles . . . 30
2.1.2 The stabilizer formalism . . . 30
2.1.2.1 Stabilizer states . . . 31
2.1.2.2 Stabilizer groups . . . 31
2.1.2.3 Stabilizer codes . . . 31
2.1.2.4 Error detection mechanism . . . 32
2.1.2.5 Logical gates . . . 33
2.1.2.6 Example: the Steane code . . . 34
2.1.3 The surface code . . . 35
2.1.3.1 Stabilizers . . . 36
2.1.3.2 Logical operators . . . 37
2.1.3.3 Decoding . . . 37
2.2 Fault-tolerant quantum computing . . . 38
2.2.1 Lattice surgery . . . 39
2.2.1.1 Merge operation . . . 39
2.2.1.2 Split operation . . . 41
2.2.2 Magic state distillation . . . 42
2.2.2.1 Magic states . . . 43
2.2.2.2 Triorthogonal codes . . . 44

3 The ZX-Calculus 47
3.1 ZX-diagrams . . . 48
3.1.1 Nodes and wires . . . 48
3.1.2 Spiders . . . 48
3.1.3 Structure of ZX-diagrams . . . 48
3.2 Rules of the ZX-calculus . . . 49

3.2.1 Only connectivity matters . . . 49
3.2.2 Identity rule . . . 49
3.2.3 Spider fusion . . . 50
3.2.4 Copy and bialgebra rules . . . 50
3.2.5 Node transformation rules . . . 51
3.3 Quantum gates in the ZX-calculus . . . 52
3.3.1 From quantum circuits to ZX-diagrams . . . 52
3.3.2 Phase and Pauli gadgets . . . 53
3.4 The ZX-calculus as a language for surface code lattice surgery . . . 54
3.4.1 Lattice surgery operations in ZX-calculus . . . 54
3.4.2 Pauli Fusion flow . . . 55

II Quantum Circuit Synthesis of Arithmetic Operators 59

4 Ancilla-Free Quantum Adder with Polylogarithmic Depth 61
4.1 Logarithmic-depth implementation of the CNOT ladder operator . . . 62
4.2 Polylogarithmic-depth implementation of the Toffoli ladder operator . . . 67
4.3 Application to quantum addition . . . 72
4.4 Extension to controlled addition . . . 74

5 Quantum Binary Field Multiplication with Low Space-Time Cost 79
5.1 Binary field multiplication . . . 80
5.2 Synthesis algorithm for binary field multiplication . . . 82
5.2.1 Subquadratic Toffoli gate count . . . 82
5.2.2 Linear depth . . . 87
5.3 Logarithmic depth multiplication for particular families of primitive polynomials 89
5.3.1 Trinomials . . . 89
5.3.2 Equally spaced polynomials . . . 90

III Quantum Circuit Optimization 93

6 Optimal Hadamard Gate Count for Clifford+R_Z Synthesis of Pauli Rotation Sequences 95
6.1 Problem statement . . . 97
6.1.1 Pauli rotation sequences . . . 97
6.1.2 Diagonalization network . . . 98
6.2 Hadamard gate minimization . . . 99

6.2.1 Diagonalization network synthesis algorithm . . . 99
6.2.1.1 Complexity analysis . . . 100
6.2.1.2 Hadamard gate count . . . 101
6.2.1.3 Diagonalization network synthesis example . . . 102
6.2.2 Optimality . . . 104
6.2.2.1 Pauli rotations ordering . . . 108
6.2.2.2 Other gate sets . . . 108
6.2.3 Extension to Clifford+R_Z circuit re-synthesis . . . 108
6.3 Internal Hadamard gate minimization . . . 110
6.3.1 Algorithm . . . 110
6.3.2 Optimality . . . 111
6.4 Improving the complexity . . . 115
6.4.1 H-Opt algorithm . . . 118
6.4.2 Internal-H-Opt algorithm . . . 119
6.5 Benchmarks . . . 121
6.5.1 Benchmarks analysis . . . 123
6.5.2 Outlook . . . 123

7 Angle-Agnostic Optimization of Non-Clifford Rotation Gates 125
7.1 Pauli rotation merging . . . 126
7.1.1 Problem statement . . . 126
7.1.2 Lowering the number of commutativity checks . . . 127
7.2 Extension for Clifford+T circuits . . . 131
7.3 Benchmarks . . . 133

8 Optimality in Parametrized Clifford Circuits with Non-Repeated Parameters 137
8.1 Parametrized Clifford circuits . . . 138
8.2 Optimality in the number of parametrized rotations . . . 139
8.3 Optimality in the number of Hadamard gates and internal Hadamard gates . 146

9 T-Count Optimization 149
9.1 Problem statement . . . 151
9.1.1 T-count optimization in Hadamard-free circuits . . . 151
9.1.2 Weighted polynomial and signature tensor . . . 152
9.1.3 Phase polynomial and parity table . . . 154
9.2 T-count reduction algorithms . . . 157
9.2.1 Third order homogeneous polynomials elimination algorithm . . . 157

9.2.2 Improving the TODD algorithm . . . 161
9.3 Benchmarks . . . 171
9.3.1 TOHPE and FastTODD algorithms . . . 171
9.3.1.1 With ancillary qubits . . . 173
9.3.1.2 Without ancillary qubits . . . 173
9.3.2 Galois field multiplier circuits . . . 173
9.4 Extension to higher levels of the Clifford hierarchy . . . 176
9.4.1 Symmetric tensor rank decomposition problem . . . 176
9.4.2 $R_Z(\pi/2^d)$-count upper bound in $\{\text{CNOT}, R_Z(\pi/2^d), R_Z(2\pi/2^d)\}$ circuits 179
9.4.3 Generalization of the FastTODD algorithm for optimizing $R_Z(\pi/2^d)$ gates 182
9.5 $R_Z(\pi/2^d)$-count upper bound in universal gate sets . . . 186

IV Optimization of Fault-Tolerant Quantum Computing using ZX-Calculus and Graph Theory 189

10 Hadamard Gate Degadgetization 191
10.1 Problem statement . . . 192
10.2 Hadamard gate degadgetization algorithm . . . 193
10.3 NP-hardness . . . 193

11 Qubit-Count Optimization using ZX-Calculus 195
11.1 Qubit-count optimization for lattice surgery using ZX-calculus . . . 196
11.1.1 Qubit-count optimization via spider ordering . . . 196
11.1.2 Qubit-count optimization using the fusion rule . . . 200
11.2 Qubit-count optimization in quantum circuits using ZX-calculus . . . 206
11.3 Benchmarks . . . 207
11.4 Perspectives . . . 207
11.4.1 Correction strategy and depth optimization . . . 209
11.4.2 Qubit-count optimization using a complete ZX-calculus equational theory . . . 209

Conclusion 211

Personal Bibliography 215

Bibliography 217

List of Figures

0 Compilation stack formed by the results presented in this thesis. 6

1.1 The Bloch sphere. 14
1.2 Hadamard gate gadgetization circuit. 23

2.1 Illustration of the [[7, 1, 3]] Steane code. 34
2.2 Surface code layout for a single logical qubit of size 5×5. 36
2.3 Quantum circuits for stabilizer measurements in the surface code. 37
2.4 Example of surface code error decoding on a distance-5 surface code logical qubit. 39
2.5 Example of a lattice surgery rough merge operation between two surface code logical qubits. 41
2.6 Example of a lattice surgery rough split operation between two surface code logical qubits. 43
2.7 Teleportation circuit to implement a T gate using a magic state. 44
2.8 Quantum circuit for the 15-to-1 magic state distillation protocol. 45

3.1 Translation table for quantum circuits and ZX-diagrams. 52
3.2 Hadamard gate gadgetization in ZX-calculus. 53
3.3 A graph-like ZX-diagram $\mathcal{D}$ and its associated signature graph $\mathcal{G}_{\mathcal{D}}$. 56

4.1 CNOT circuit produced by Algorithm 1 to implement the $\mathcal{L}_1^{(9)}$ operator. 64
4.2 Multi-controlled NOT circuit produced by Algorithm 2 to implement the $\mathcal{L}_2^{(9)}$ operator. 69
4.3 Quantum ripple-carry adder where $n = 5$. 73

5.1 Linear depth multiplication circuit for $GF(2^4)$ with primitive polynomial $P(x) = x^4 + x + 1$. 88

6.1 Operations associated to the conjugation of a sequence of Pauli products by Clifford gates. 97
6.5 Construction example of C' and $\mathcal{S}'$ for the proof of Theorem 9. 106
6.6 Execution example of the diagonalization network synthesis algorithm. 112

7.1 Example of a circuit for which Algorithm 10 is suboptimal. 131

9.1 Example of classically controlled Clifford gates resulting from the gadgetization of a Hadamard gate. 153
9.2 An implementation of the U_f gate with weighted polynomial $f(\boldsymbol{x}) = 4x_1x_2x_3$, which corresponds to the CCZ gate. 154

9.3 Overview of the process for the optimization of the number of T gates in Hadamard-free circuits. 155

10.1 Example demonstrating the procedure used for degadgetizing a maximal number of Hadamard gates in a quantum circuit. 194

11.1 Example demonstrating how rearranging the order of spiders in a given ZX-diagrams can reduce the number of logical qubits needed for its implementation. 197
11.2 Example demonstrating the procedure used for finding an optimal arrangement of the spiders within a given ZX-diagram to minimize the number of logical qubits required for its implementation via lattice surgery operations. 199
11.3 Example demonstrating how the maximum number of wires in any vertical cut of a given ZX-diagram can be optimized so that it is equal to $n + 1$, where n is the number of spiders in the initial ZX-diagram. 201
11.4 Example demonstrating the procedure used for optimizing the number of qubits required to implement a given ZX-diagram using lattice surgery operations by rearranging its spiders and using the spider fusion rule. 204
11.5 Example of ZX-diagrams that are requiring $pw(\mathcal{D})$, $pw(\mathcal{D}) + 1$ and $pw(\mathcal{D}) + 2$ logical qubits respectively to be implemented through lattice surgery operations. 205
11.6 Overview of the process described for optimizing the number of qubits in a quantum circuit. 206

List of Tables

4.1 Comparison of state-of-the-art quantum adders in terms of ancillae, depth, and gate count. 62

5.1 Toffoli-count, qubit-count, depth and space-time cost comparison between different quantum circuits performing multiplication over $GF(2^n)$. 80

6.1 Benchmark for the optimization of the number of internal Hadamard gates. . . . 122

7.1 Comparison of different procedures for optimizing the number of T gates in Clifford+T circuits. 134

9.1 Benchmark for the optimization of the number of T gates. 172
9.2 Benchmark for the optimization of the number of T gates without ancillary qubits. 174
9.3 Benchmark for the optimization of the number of T and CCZ gates in GF(2^n) multiplier circuits. 175

11.1 Qubit-count achieved by different methods on a set of quantum circuits with an optimized number of T gates. 208

List of Algorithms

1 Logarithmic-depth CNOT circuit synthesis for $\mathcal{L}_1^{(n-1)}$ . . . 63
2 Logarithmic-depth MCX circuit synthesis for $\mathcal{L}_{\boldsymbol{\alpha}}$ . . . 68
3 Ancilla-free ripple-carry adder . . . 73
4 Ancilla-free controlled ripple-carry adder . . . 74

5 Diagonalization network synthesis with a minimal number of H gates . . . 100
6 Diagonalization network synthesis with a minimal number of internal H gates . . . 111
7 H-Opt . . . 116
8 Internal-H-Opt . . . 117

9 Computes the rank vector associated with a Clifford+R_Z circuit . . . 128
10 Black box rotation merging algorithm . . . 129
11 Non-Clifford rotation merging algorithm . . . 132

12 Grouping of Pauli rotations . . . 151
13 Third order homogeneous polynomials elimination algorithm . . . 161
14 Faster version of the third order duplicate and destroy algorithm . . . 169

15 Optimize a ZX-diagram using the fixed-endbags pathwidth problem . . . 203

Introduction

Whether classical or quantum, a computer relies on two fundamental components: hardware and software. These elements form a symbiotic relationship, each dependent on the other for the system to function effectively. Hardware provides the physical infrastructure, comprising processors, memory units, storage devices, and input-output interfaces. It establishes the foundational capabilities and limitations of the system. The architecture of the hardware determines the basic operations that can be performed and the speed at which they can be executed. Software, on the other hand, is the intangible component that harnesses the potential of the hardware. It consists of programs, data, and instructions that direct the hardware's operations. Software ranges from low-level machine code, which interacts directly with the hardware, to high-level programming languages that abstract the complexities of hardware operations.

Central to the hardware-software interaction is the process of compilation. Compilation serves as a crucial bridge between the high-level abstractions of software and the low-level operations executable on the hardware. It is the process by which human-readable source code is transformed into machine-executable instructions. In classical computing, compilers have been refined over decades to optimize code for various architectures, balancing factors such as execution speed, memory usage, and power consumption. In contrast, compilation for quantum computing is still in its early stages. While the fundamental goal of translating high-level code into machine instructions remains, quantum computing presents unique challenges and opportunities that require a radical rethinking of compilation strategies and the development of new methods. We are, in essence, tasked with building an entirely new and innovative compilation toolchain from the ground up.

Creating a compilation toolchain for quantum computing notably involves developing new programming languages, designing quantum-specific intermediate representations, and creating novel optimization techniques that account for the peculiarities of quantum systems. Another critical aspect of quantum compilation is the integration of error correction techniques and fault-tolerant protocols. These are essential to counteract the effects of decoherence and quantum noise, thereby making quantum computation more reliable and robust. Additionally, quantum compilers should ideally be adaptable to the rapidly evolving quantum hardware architectures and capable of exploiting the unique advantages offered by different hardware technologies. Addressing these challenges is crucial for bridging the gap between theoretical quantum algorithms and their actual execution on quantum hardware. As such, compilation plays a pivotal role in assessing the cost of fault-tolerant quantum computing and improving its feasibility and practicality.

This thesis tackles some of these compilation challenges in the context of large-scale quantum computing. We present efficient algorithms, automated procedures, and formal methods for optimizing fault-tolerant quantum computing. To achieve this, we utilize a wide range of models for representing quantum computation, spanning from abstract formalisms to low-level representations. These models include high-level representations of some quantum operators, the quantum

circuit model, and the ZX-calculus, a rigorous graphical language developed for reasoning about quantum processes. Using these diverse representations allows us to design optimization strategies that target different levels of abstraction in the quantum compilation pipeline, thereby contributing to the development of a comprehensive and performant compilation toolchain for fault-tolerant quantum computing.

The results of this thesis are presented in the order in which they would intervene in this compilation toolchain, following the natural flow from high-level quantum operations to hardware-executable instructions. The first step considered is quantum circuit synthesis. A quantum circuit is a model for quantum computation that represents quantum operations as a sequence of elementary operations called quantum gates and measurements. These quantum gates are the building blocks of quantum computation, analogous to classical logic gates but operating on quantum bits (qubits). While some classical gates, like AND and OR, perform irreversible operations on classical bits, quantum gates are unitary operations that reversibly transform quantum states. They can create and manipulate quantum phenomena such as superposition and entanglement, which have no classical counterparts. The process of quantum circuit synthesis consists in translating abstract quantum operations, often expressed in mathematical form, into sequences of elementary quantum gates. The synthesis phase establishes the baseline implementation that will serve as input to subsequent optimization procedures.

The choice of quantum gates available for synthesis, known as the gate set, significantly impacts both the synthesis process and the resulting circuit implementation. Common elementary gate sets include single-qubit rotations and two-qubit controlled operations. A gate set is called universal if any unitary operation can be exactly implemented using a finite sequence of gates from this set. However, it turns out that any finite gate set that gives rise to a finite set of unitary operations cannot be universal. This leads to the notion of approximate universality: a gate set is approximately universal if any unitary operation can be approximated to arbitrary precision using a finite sequence of gates from this set. This notion is sufficient for practical purposes, as we can always choose an approximation precision that makes any experimental differences undetectable.

The Solovay-Kitaev theorem, a fundamental result in quantum computation, ensures that any quantum gate can be approximated to arbitrary precision using a finite approximately universal gate set. Moreover, the theorem provides an explicit algorithm for finding such approximations with a polylogarithmic overhead in the number of gates required. This theorem guarantees that we can, in principle, implement any quantum algorithm using only a small set of elementary gates. However, the choice of this approximately universal gate set has profound practical implications, particularly in the context of quantum error correction and fault-tolerant quantum computing.

Quantum systems are inherently fragile and highly susceptible to environmental noise and operational imperfections. To address this fundamental challenge, quantum error-correcting codes have been developed to protect quantum information by encoding it across multiple physical qubits, forming what we call logical qubits. While error correction is essential, it is insufficient on its own. We must also perform reliable quantum operations on these logical qubits while maintaining the code's error-correcting properties, a characteristic known as fault tolerance.

In quantum error-correcting codes, only a restricted set of gates can be implemented fault-tolerantly. Some gates, called transversal gates, can be implemented directly on encoded logical qubits while preserving the error-correcting properties of the code. Clifford operations, generated by the two-qubit controlled-NOT (CNOT) gate and single-qubit Hadamard (H) and phase (S) gates, often admit efficient fault-tolerant implementations, for example through transversal operations. However, Clifford gates alone are insufficient for universal quantum computation. To achieve universality, Clifford gates must be supplemented with at least one non-Clifford gate.

A common choice is the single-qubit T gate, leading to the widely-used Clifford+T gate set. The fault-tolerant implementation of non-Clifford gates such as the T gate typically requires more complex protocols involving state preparation, measurement, and classical feedback, making them more resource-intensive than Clifford gates.

This fundamental constraint on the set of gates used to achieve fault tolerance, combined with the disparity in implementation costs between different types of gates, implies that the synthesis problem cannot be evaluated solely on the total gate count. Instead, we must consider both the fault-tolerant implementability of gates in our target error-correcting code and their respective implementation costs.

Beyond gate counts, several additional metrics can guide the synthesis process. The circuit depth, corresponding to the length of the longest path through the circuit, directly impacts execution time. The number of qubits required represents a fundamental constraint that determines the physical resources needed to execute the circuit on quantum hardware. The overall space-time cost combines these factors to estimate total resource requirements. Synthesis algorithms must therefore balance multiple, sometimes competing, optimization objectives while working with a restricted gate set.

Once the synthesis of the different parts of a quantum algorithm has been performed, we obtain a quantum circuit implementation. The next phase of the quantum compilation toolchain considered in this thesis is quantum circuit optimization. Circuit optimization takes the output of the synthesis phase and transforms it into an equivalent circuit that minimizes specific cost metrics while preserving its functionality.

Quantum circuit optimization can be approached from multiple angles. A common method is to rely on quantum circuit equalities and pattern matching to apply a series of local circuit transformations. This approach identifies specific sequences of gates that can be replaced by more efficient equivalent sequences. For instance, adjacent CNOT gates acting on the same qubits cancel out, and consecutive single-qubit rotation gates can often be combined or simplified. While individual rewriting rules are typically straightforward, their repeated application or combination creates a vast space of possible transformations.

Given the exponential growth of the transformation space, exhaustive search is generally impractical even for moderately-sized quantum circuits. This motivates the development of efficient strategies for navigating the transformation space effectively. One approach involves analyzing the circuit to design customized rewriting rules specific to its structure. Rather than relying solely on a fixed set of identities, this methods creates transformations that are tailored to the particular circuit or category of circuits being optimized. For instance, this approach has been used in state-of-the-art algorithms to reduce the number of T gates in Clifford+T circuits. By focusing on transformations likely to be relevant for the given quantum circuit, these methods can dramatically reduce the search space and achieve better optimization results in reasonable computational time.

An alternative approach for optimizing quantum circuits is to do a resynthesis of some parts of the quantum circuit. Rather than applying local rewriting rules, resynthesis first extracts a higher-level description of the operation performed by a section of the circuit. This extracted operation is then synthesized anew using specialized methods that target the desired optimization metrics. A key advantage of resynthesis lies in its potential to overcome local optima by completely restructuring the circuit. Since the high-level descriptions of the extracted parts are typically invariant under local circuit transformations, this optimization method is less constrained by the initial circuit structure.

The choice of the optimization approach depends on multiple factors including the specific optimization targets, characteristics of the input circuit such as its structure or size, and available

computational resources. Additionally, the interaction between different optimization metrics adds another layer of complexity to the optimization process. Notably, the order in which different optimization metrics are addressed can significantly impact the final results. For example, in Clifford+T circuits, optimizing the number of Hadamard gates before minimizing T-count has been shown to lead to better overall results, both in terms of T-count and ancillary qubit requirements. As such, studying the interdependencies between different metrics is essential for designing an effective optimization strategy.

Quantum circuit optimization can significantly reduce resource requirements, but the optimized circuit must then be translated into a fault-tolerant implementation which presents additional challenges. The quantum circuit model, though valuable for algorithm design and analysis, has inherent limitations when representing fault-tolerant operations at the logical level. In particular, its gate-by-gate approach and unitary properties fail to accurately capture the measurement-based nature of common fault-tolerant protocols. These limitations motivate the development and adoption of alternative representations that extend beyond the traditional quantum circuit model.

The ZX-calculus has emerged as a particularly useful representation to address these challenges. Based on tensor networks, this graphical language represents quantum operations as diagrams with well-defined rewrite rules that preserve semantic equivalence. Similar to how circuit identities enable circuit transformations, these rewrite rules allow rigorous manipulation of ZX-diagrams. However, the ZX-calculus offers greater flexibility through its fundamental building blocks (the green and red spiders) which can represent more elementary operations than quantum gates. Notably, these building blocks of the ZX-calculus have a natural correspondence with some fault-tolerant operations. In particular, it has been demonstrated that the ZX-calculus is a language for surface code lattice surgery, a measurement-based approach for performing fault-tolerant operations between encoded quantum states. Where the circuit model would require more complex sequences of gates to represent such operations, the ZX-calculus can capture them more directly and intuitively.

The ZX-calculus can also be used to establish relations between different computational models and optimization problems. For instance, as we demonstrate in this thesis, optimizing the qubit-count across various quantum computational models can be effectively reformulated as graph-theoretical problems by relying on the ZX-calculus. Such connection provides access to established algorithmic techniques and theoretical results from graph theory, offering new perspectives on some optimization problems.

Moreover, the ZX-calculus can be used as an intermediate representation for optimizing quantum circuits. A quantum circuit can be translated into a ZX-diagram, optimized using the ZX-calculus rewrite rules, and then translated back into a circuit. This transformation process can reveal optimization opportunities that are not readily apparent in the standard quantum circuit model.

Ultimately, as outlined in this introduction, the chosen approach and representation model depend on multiple factors, such as the selected quantum error-correcting code, the computational model used to achieve fault-tolerant quantum computing, the optimization metrics, the available computational resources and time constraints, and the level of abstraction. In this thesis we make use of the strengths of these diverse approaches and representations to develop new results for quantum compilation, with a particular focus on optimizing resources for fault-tolerant quantum computing.

Contributions and organization of the manuscript. This manuscript is structured in four parts. Part I establishes the theoretical foundations necessary for understanding the contributions presented in this manuscript:

- Chapter 1 provides an introduction to quantum computing and the quantum circuit model, presenting the key concepts and notations used throughout this manuscript.
- Chapter 2 presents fundamental principles for quantum error correction and fault-tolerant quantum computing, with a focus on surface code lattice surgery.
- Chapter 3 introduces the ZX-calculus, a graphical language used in this thesis to formulate and solve optimization problems.

The contributions of this thesis are presented in the subsequent parts. Figure 0 illustrates how the results presented in each chapter contribute to different stages of the quantum compilation pipeline. Part II focuses on quantum circuit synthesis of arithmetic operators, which are fundamental building blocks for many quantum algorithms:

- Chapter 4 introduces a novel quantum adder circuit achieving polylogarithmic depth without ancillary qubits.
- Chapter 5 presents an efficient quantum multiplier for binary fields, with a subquadratic number of T gates and a low space-time cost. This operation plays a crucial role in quantum cryptanalysis of binary elliptic curve cryptography.

Part III focuses on quantum circuit optimization:

- Chapter 6 presents an algorithm for the synthesis of Pauli rotation sequences over the Clifford+R_Z gate set with a minimal number of Hadamard gates. This optimization is particularly useful as a pre-processing step for optimizing the number of T gates in quantum circuits.
- Chapter 7 introduces a fast and efficient algorithm for optimizing the number of non-Clifford single-qubit rotation gates without requiring knowledge of rotation angles. This approach notably uses the Hadamard gate optimization algorithm presented in Chapter 6.
- Chapter 8 establishes the optimality of the algorithms presented in Chapters 6 and 7 for parameterized Clifford circuits with non-repeated parameters. Parameterized quantum circuits are quantum circuits that include adjustable parameters in their quantum gates. Such circuits arise naturally in variational quantum algorithms.
- Chapter 9 presents algorithms for optimizing the number of T gates in quantum circuits. We demonstrate that our algorithms provide a lower T-count on most quantum circuits evaluated while being much faster than previous methods. We also prove a new constructive upper bound on the number of T gates in Clifford+T circuits.

Part IV presents optimization techniques for fault-tolerant quantum computing using ZX-calculus and graph theory:

- Chapter 10 introduces a new method for optimizing the number of qubits in quantum circuits based on Hadamard gate degadgetization. We prove that this optimization problem is equivalent to the well-known feedback vertex set problem in graph theory.

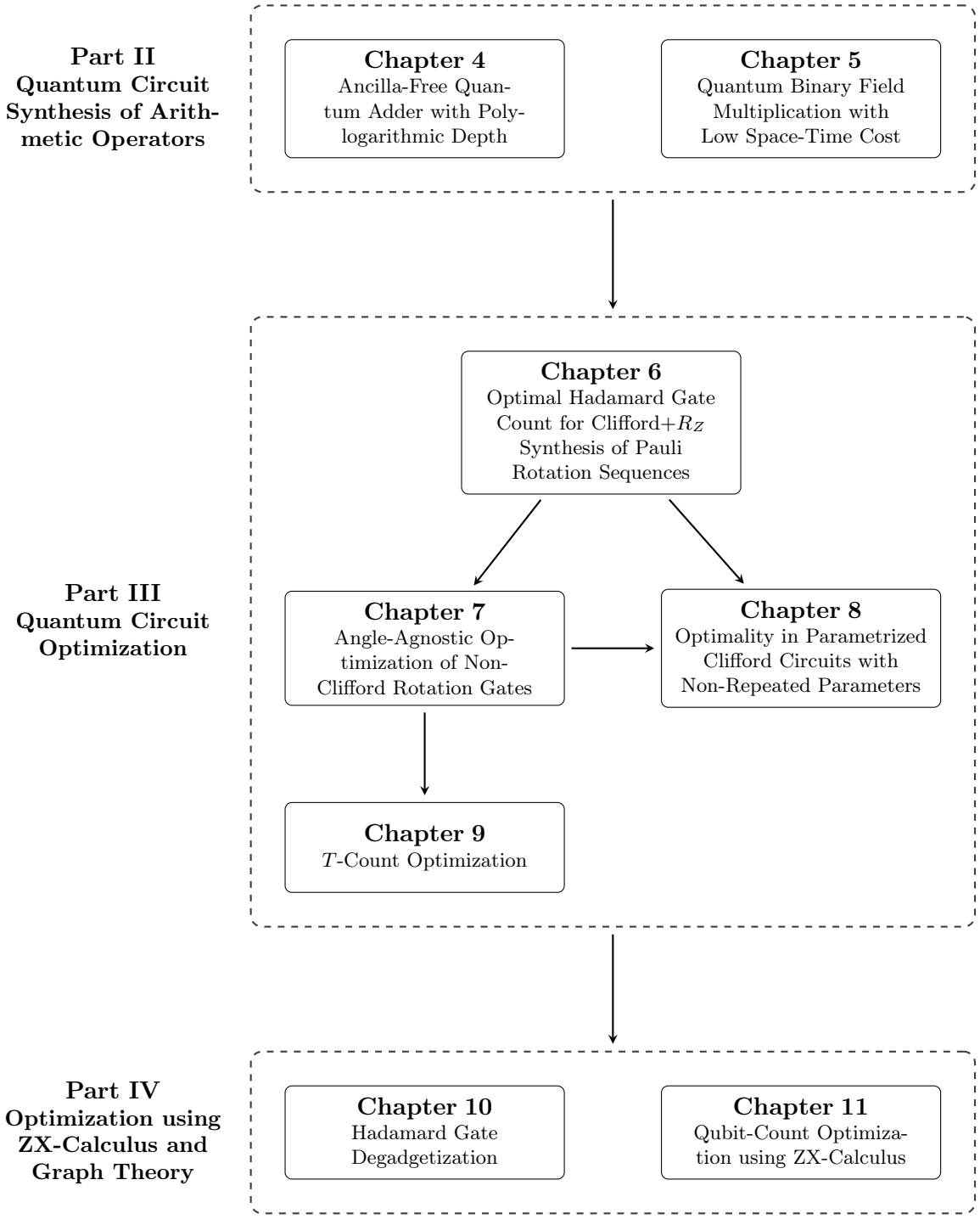


Figure 0: Compilation stack formed by the results presented in this thesis.

- Chapter 11 presents a framework for optimizing the number of qubits in various computational models by relying on the ZX-calculus. We show how the qubit-count optimization problem relates to well-known graph theoretical problems such as the minimal-width path-decomposition problem.

The results presented in each chapter are based on the following articles:

- Chapter 4: [1] Maxime Remaud, and Vivien Vandaele. Ancilla-free Quantum Adder with Sublinear Depth. *arXiv preprint*, 2025. arXiv:2501.16802.
- Chapter 5: [2] Vivien Vandaele. Quantum binary field multiplication with subquadratic Toffoli gate count and low space-time cost. *arXiv preprint*, 2025. arXiv:2501.16136.
- Chapter 6: [3] Vivien Vandaele, Simon Martiel, Simon Perdrix, and Christophe Vuillot. Optimal Hadamard Gate Count for Clifford+T Synthesis of Pauli Rotations Sequences. *ACM Transactions on Quantum Computing*, 5(1):1–29, February 2024. arXiv:2302.07040. doi:10.1145/3639062.
- Chapters 7 and 8: [4] Vivien Vandaele, Simon Perdrix, and Christophe Vuillot. Optimal number of parametrized rotations and Hadamard gates in parametrized Clifford circuits with non-repeated parameters. *arXiv preprint*, 2024. arXiv:2407.07846.
- Chapter 9: [5] Vivien Vandaele. Lower T-count with faster algorithms. *arXiv preprint*, 2024. arXiv:2407.08695.
- Chapters 10 and 11: [6] Vivien Vandaele. Qubit-count optimization using ZX-calculus. *arXiv preprint*, 2024. arXiv:2407.10171.

Part I

Background

Chapter 1

Quantum Computing and the Quantum Circuit Model

Quantum computing exploits quantum mechanical phenomena to manipulate information and perform computation in ways fundamentally different from classical approaches. The field emerged from pioneering ideas in the 1980s, when scientists such as Paul Benioff [10], Richard Feynman [11] and Yuri Mani [12] recognized that quantum systems could potentially be used to solve computational problems in novel ways.

The development of quantum computing gained momentum in the 1990s with the discovery of algorithms that showcased its potential power. Early algorithms like the Deutsch-Jozsa algorithm [13] and Simon's algorithm [14] demonstrated that quantum computers could be used to solve certain problems more efficiently than classical computers. While these algorithms addressed somewhat artificial problems, they provided crucial insights into the potential advantages of quantum computation. One landmark achievement was Peter Shor's algorithm [15], published in 1994, which demonstrated that a quantum computer could factor large numbers in polynomial time, providing a superpolynomial speedup over the best-known classical algorithms. This discovery had profound implications for cryptography, as it showed that quantum computers could potentially break widely-used public-key cryptographic systems that rely on the assumed difficulty of factoring large numbers, such as RSA encryption. Further algorithmic breakthroughs followed such as Lov Grover's algorithm [16], which demonstrated a quadratic speedup for unstructured search problems. These early algorithmic breakthroughs established that quantum computers could offer computational advantages over classical machines for certain important problems, spurring both theoretical research and experimental efforts to build practical quantum devices.

To describe and implement quantum algorithms, researchers needed a systematic model for representing quantum computations. The quantum circuit model has emerged as the standard framework for this purpose. This model represents quantum computations as sequences of quantum gates acting on qubits, providing a structured approach for designing, analyzing, and implementing quantum algorithms. A given quantum algorithm can be implemented by various quantum circuits, each with different trade-offs and implementation costs. This gives rise to various problems in quantum circuit synthesis, the process of translating quantum operators into efficient sequences of implementable quantum gates.

This chapter provides an introduction to quantum computing and the quantum circuit model. These foundational notions are then used to describe various quantum circuit synthesis problems, several of which are addressed in this thesis.

1.1 Introduction to quantum computing

This section introduces fundamental concepts and mathematical tools for describing and manipulating quantum information. We begin by presenting key mathematical notations, such the Dirac notation. Then, we describe quantum states, which serve as the fundamental representation of quantum information, with the qubit being the basic unit. Finally, we demonstrate how quantum computation is achieved through the manipulation of quantum states via quantum operators and measurements.

1.1.1 Mathematical framework and notations

1.1.1.1 Dirac notation

To effectively describe and manipulate quantum-specific concepts such as qubits, superposition, entanglement, and high-dimensional vector spaces, we need a notation system that is formal, concise, and intuitive. The Dirac notation (also known as the bra-ket notation) was designed for this purpose and has become the standard notation in quantum mechanics for linear algebra and linear operators on complex vector spaces.

The Dirac notation is built upon two fundamental objects: kets and bras. A ket, denoted as $|\psi\rangle$, represents a column vector in a complex vector space. For example, in a two-dimensional space, we have:

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{1.1}$$

where α and β are complex numbers. A bra, denoted as $\langle\psi|$, is the conjugate transpose (denoted by $\dagger$) of a ket. It is represented as a row vector, for example:

$$\langle\psi| = |\psi\rangle^\dagger = \begin{pmatrix} \alpha^* & \beta^* \end{pmatrix} \tag{1.2}$$

where $*$ denotes complex conjugation.

The inner product of two vectors $|\psi\rangle$ and $|\phi\rangle$ is denoted as $\langle\psi|\phi\rangle$. For example:

$$\langle\psi|\phi\rangle = \langle\psi|\,|\phi\rangle = \begin{pmatrix} \alpha^* & \beta^* \end{pmatrix} \begin{pmatrix} \gamma \\ \delta \end{pmatrix} = \alpha^*\gamma + \beta^*\delta. \tag{1.3}$$

And the outer product of two ket vectors $|\psi\rangle$ and $|\phi\rangle$ is denoted as $|\psi\rangle\langle\phi|$. For example:

$$|\psi\rangle\langle\phi| = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \begin{pmatrix} \gamma^* & \delta^* \end{pmatrix} = \begin{pmatrix} \alpha\gamma^* & \alpha\delta^* \\ \beta\gamma^* & \beta\delta^* \end{pmatrix}. \tag{1.4}$$

In this chapter, we will heavily rely on the Dirac notation to formalize and easily manipulate quantum states and operators.

1.1.1.2 The computational, diagonal, and circular bases

In quantum computing, we often represent ket vectors with one of the following orthonormal bases: the computational basis, the diagonal basis, and the circular basis.

The two-dimensional computational basis consists of two orthonormal vectors denoted by $|0\rangle$ and $|1\rangle$. These are represented as:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{1.5}$$

In a two-dimensional space, a ket can be decomposed in the computational basis as follows:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix} \tag{1.6}$$

where α and β are complex numbers.

The diagonal basis (also known as the Hadamard basis) is formed by the kets $|+\rangle$ and $|-\rangle$, defined as follows:

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\ 1\end{pmatrix}, \quad |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\ -1\end{pmatrix}. \tag{1.7}$$

Finally, the circular basis is formed by the kets $|i\rangle$ and $|-i\rangle$, defined as follows:

$$|i\rangle = \frac{1}{\sqrt{2}}(|0\rangle + i\,|1\rangle) = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\ i\end{pmatrix}, \quad |-i\rangle = \frac{1}{\sqrt{2}}(|0\rangle - i\,|1\rangle) = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\ -i\end{pmatrix}. \tag{1.8}$$

1.1.1.3 Combining ket vectors

The tensor product, denoted by $\otimes$, is a way of combining vector spaces to form larger vector spaces. For two vectors $|\psi\rangle$ and $|\phi\rangle$, their tensor product is defined as:

$$|\psi\rangle \otimes |\phi\rangle = \begin{pmatrix}\psi_1\,|\phi\rangle\\ \psi_2\,|\phi\rangle\\ \vdots\\ \psi_{2^n}\,|\phi\rangle\end{pmatrix} \tag{1.9}$$

where ψ_i are the elements of the ket vector $|\psi\rangle$.

For simplicity and readability, it is common to omit the $\otimes$ symbol when writing tensor products of ket vectors:

$$|\psi\rangle \otimes |\phi\rangle = |\psi\rangle\,|\phi\rangle = |\psi\phi\rangle\,. \tag{1.10}$$

Also, we will use the notation $|\psi\rangle^{\otimes n} = |\psi\rangle^n$ to represent the tensor product of n ket vectors $|\psi\rangle$. For example:

$$|0\rangle^{\otimes 3} = |0\rangle^3 = |0\rangle \otimes |0\rangle \otimes |0\rangle = |0\rangle\,|0\rangle\,|0\rangle = |000\rangle\,. \tag{1.11}$$

The two-dimensional computational, diagonal and circular bases, described in Section 1.1.1.2, can be naturally extended to higher dimensions using the tensor product. For example, for a space of dimension 2^n, the computational basis vectors are:

$$|x_1x_2...x_n\rangle = |x_1\rangle \otimes |x_2\rangle \otimes \ldots \otimes |x_n\rangle \tag{1.12}$$

where $x_i \in \{0, 1\}$ for all i, and $|0\rangle$ and $|1\rangle$ are the two-dimensional basis states as defined in Section 1.1.1.2.

We will often use the term register to denote a ket vector. This notation allows us to clearly identify and manipulate different parts of a composite system. For example, $|\psi\rangle_A\,|\phi\rangle_B$ is composed of the register A and register B.

1.1.2 Quantum states

1.1.2.1 Qubit

A qubit is the fundamental unit of quantum information, analogous to a bit in classical computing. Unlike classical bits, which can only be in one of two states (0 or 1), a qubit can exist in a superposition of both states simultaneously.

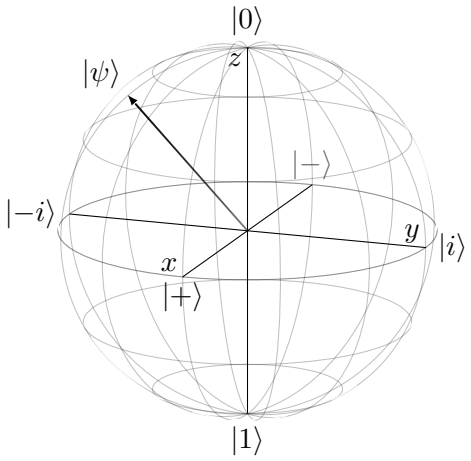


Figure 1.1: The Bloch sphere.

Pure and mixed states. Mathematically, a qubit in a pure quantum state can be represented in the computational basis by a two-dimensional ket vector of unit length:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \quad \alpha, \beta \in \mathbb{C}, \quad |\alpha|^2 + |\beta|^2 = 1. \tag{1.13}$$

The complex coefficients α and β are called probability amplitudes.

While pure states represent the most precise knowledge we can have about a quantum system, in reality, we often deal with statistical mixtures of pure states, known as mixed states. These are represented by a density matrix ρ:

$$\rho = \sum_i p_i\,|\psi_i\rangle\,\langle\psi_i| \tag{1.14}$$

where p_i are probabilities satisfying $\sum_i p_i = 1$, and $|\psi_i\rangle$ are pure states. For a pure state $|\psi\rangle$, the density matrix is simply $\rho = |\psi\rangle\,\langle\psi|$.

While mixed states are critical in understanding the practical implications of noise and decoherence in quantum systems, the theoretical framework of quantum algorithms and quantum compilation is primarily rooted in the properties of pure quantum states. Therefore, we will mainly work with pure quantum states in this thesis.

Bloch sphere representation. The state of a qubit can be visualized as a point on or inside a unit sphere in $\mathbb{R}^3$, known as the Bloch sphere. Any pure state of a qubit corresponds to a point on the surface of this sphere and can be written as:

$$|\psi\rangle = \cos(\theta/2)\,|0\rangle + e^{i\phi}\sin(\theta/2)\,|1\rangle \tag{1.15}$$

where θ and ϕ are the spherical coordinates of the point. Mixed states correspond to points inside the Bloch sphere, with the maximally mixed state at the center. The Bloch sphere representation is particularly useful for visualizing single-qubit operations, which correspond to rotations of the state vector on the sphere. Figure 1.1 provides an illustration of the Bloch sphere.

1.1.2.2 Multi-qubit quantum states

Similarly to single-qubit pure quantum states, an n-qubit pure quantum state can be represented by a 2^n-dimensional ket vector of unit length:

$$|\psi\rangle = \sum_{k\in\{0,1\}^n} \alpha_k |k\rangle, \quad \alpha_k \in \mathbb{C}, \quad \sum_{k\in\{0,1\}^n} |\alpha_k|^2 = 1. \tag{1.16}$$

Here, $|k\rangle$ represents the computational basis states, as defined in Section 1.1.1.2. For example, a two-qubit state can be written as:

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle. \tag{1.17}$$

The 2^n-dimensional ket vector representing the n-qubit pure quantum state is called the state vector. The exponential growth of the state vector's size with the number of qubits is one of the fundamental reasons why we don't know any way of efficiently simulating quantum systems with classical computers.

While some multi-qubit states can be represented as tensor products of individual qubit states, not all quantum states have this property. A multi-qubit quantum state is said to be entangled if it cannot be written as a tensor product of individual qubit states. Mathematically, for a two-qubit system, a state $|\psi\rangle$ is entangled if there do not exist single-qubit states $|\phi_1\rangle$ and $|\phi_2\rangle$ such that:

$$|\psi\rangle = |\phi_1\rangle \otimes |\phi_2\rangle. \tag{1.18}$$

Quintessential examples of entangled two-qubit states are the Bell states:

$$\begin{aligned} |\Phi^+\rangle &= \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), \quad |\Phi^-\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle), \\ |\Psi^+\rangle &= \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle), \quad |\Psi^-\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle). \end{aligned} \tag{1.19}$$

These states cannot be factored into a product of two single-qubit states, demonstrating their entangled nature. Entanglement is a key resource in many quantum information protocols, enabling phenomena such as quantum teleportation.

1.1.3 Quantum operators and measurements

We now introduce two types of operations to transform quantum states and perform quantum computation: quantum operators, which manipulate quantum states, and measurements, which extract classical information from these states.

1.1.3.1 Unitary operators

Unitary operators are the quantum equivalent of reversible classical operations. A unitary operator U is a linear operator that preserves the norm of quantum states. Equivalently, U satisfies:

$$U^\dagger U = UU^\dagger = I \tag{1.20}$$

where $U^\dagger$ is the conjugate transpose of U, and I is the identity matrix.

Multiple unitary operators can be applied sequentially to a quantum state. If $U_1, U_2, ..., U_n$ are unitary operators, their product $U_n \dots U_2 U_1$ is also a unitary operator. This property allows

us to decompose complex quantum operations into sequences of simpler unitary operator. The order of application is important, as matrix multiplication is generally non-commutative.

For example, the Hadamard operator, denoted by H, is a unitary operator defined as follows

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{1.21}$$

The Hadamard gate maps the computational basis to the diagonal basis:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle \tag{1.22}$$

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle \tag{1.23}$$

The Hadamard operator is self-inverse ($H^2 = I$) and self-adjoint ($H = H^\dagger$), and therefore satisfies the unitary operator property $H^\dagger H = HH^\dagger = I$.

Tensor product of unitary operators. When working with multi-qubit systems, unitary operators can be combined using the tensor product. Let U be an $n \times n$ matrix and V be an $m \times m$ matrix. Their tensor product $U \otimes V$ is an $(nm) \times (nm)$ matrix constructed by multiplying each element of matrix U by the entire matrix V:

$$U \otimes V = \begin{pmatrix} u_{11}V & u_{12}V & \cdots & u_{1n}V \\ u_{21}V & u_{22}V & \cdots & u_{2n}V \\ \vdots & \vdots & \ddots & \vdots \\ u_{n1}V & u_{n2}V & \cdots & u_{nn}V \end{pmatrix}. \tag{1.24}$$

The tensor product allows the construction of unitary operators acting on larger multi-qubit systems from operators acting on disjoint systems. Let U and V be unitary operators, applying U and V to different quantum systems is equivalent to applying their tensor product to the joint system:

$$(U|\psi_1\rangle) \otimes (V|\psi_2\rangle) = (U \otimes V)|\psi_1\rangle|\psi_2\rangle. \tag{1.25}$$

For example, applying the Hadamard operator on two different qubits corresponds to the operator $H \otimes H$ acting on two qubits:

$$H \otimes H = \frac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}. \tag{1.26}$$

1.1.3.2 Quantum measurements

Measurement is a fundamental operation in quantum computing, allowing us to extract classical information from quantum states. Unlike unitary operations, measurements are irreversible and probabilistic. A quantum measurement is described by a collection of measurement operators $\{M_m\}$ where m represents the possible measurement outcomes. These operators satisfy the completeness equation:

$$\sum_m M_m^\dagger M_m = I. \tag{1.27}$$

Let $|\psi\rangle$ be the quantum state before measurement. The probability $p(m)$ of obtaining the outcome corresponding to m is:

$$p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle \tag{1.28}$$

After the measurement, if the outcome corresponding to m is observed, the quantum state collapses to:

$$|\psi_m\rangle = \frac{M_m|\psi\rangle}{\sqrt{p(m)}}. \tag{1.29}$$

This is the post-measurement state, normalized by the probability of the outcome. This collapse represents the irreversible nature of quantum measurement, fundamentally altering the quantum state based on the measurement result.

A particularly important type of measurement is Pauli measurements, where the measurement operators correspond to the Pauli matrices I, X, Y, and Z, or their tensor products over multiple qubits. Pauli measurements are especially significant in quantum error correction, where they are used to detect and correct errors by measuring stabilizer operators, which are typically tensor products of Pauli matrices.

Example. Consider measuring a qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ in the computational basis, corresponding to a Pauli measurement in the Z basis. The measurement operators are:

$$M_0 = |0\rangle\langle 0| = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \tag{1.30}$$

$$M_1 = |1\rangle\langle 1| = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \tag{1.31}$$

The probabilities of the outcomes are $p(0) = |\alpha|^2$ and $p(1) = |\beta|^2$, where $|\alpha|^2 + |\beta|^2 = 1$. After the measurement, the state collapses to either $|0\rangle$ or $|1\rangle$, with the respective probabilities given above.

The choice of the measurement basis can lead to different probabilistic outcomes and post-measurement states. For example, the projectors for measuring the same quantum state in the diagonal basis $\{|+\rangle, |-\rangle\}$, are:

$$M_+ = |+\rangle\langle +| = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \tag{1.32}$$

$$M_- = |-\rangle\langle -| = \frac{1}{2}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \tag{1.33}$$

The probabilities of the outcomes are:

$$p(+) = |\langle +|\psi\rangle|^2 = \left|\frac{\alpha+\beta}{\sqrt{2}}\right|^2 \tag{1.34}$$

$$p(-) = |\langle -|\psi\rangle|^2 = \left|\frac{\alpha-\beta}{\sqrt{2}}\right|^2 \tag{1.35}$$

After the measurement, the state collapses to either $|+\rangle$ or $|-\rangle$, with the respective probabilities given above.

1.2 Quantum gates and quantum circuits

This section introduces quantum gates and the quantum circuit model, which provide a practical framework for describing and implementing quantum algorithms. We begin by presenting common quantum gates, which are the elementary operations that can be performed on quantum states. Then, we describe how these gates can be combined to form quantum circuits, and we present several problems in quantum circuit synthesis and optimization.

1.2.1 Quantum gates

Quantum gates are the building blocks of quantum circuits, just like classical logic gates (AND, OR, NOT, etc.) are the building blocks of classical circuits. They are represented by unitary operators, as defined in Section 1.1.3.1, that act on qubits to transform their states. This section introduces the most common quantum gates.

1.2.1.1 Pauli gates

The Pauli gates are fundamental single-qubit gates in quantum computing, represented by the Pauli matrices:

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{1.36}$$

The Pauli matrices are self-adjoint, meaning that $P^\dagger = P$ for $P \in \{I, X, Y, Z\}$. Consequently, applying a Pauli gate twice yields the identity operation. The X, Y, and Z Pauli gates represent rotations of π radians around the x-, y-, and z-axes of the Bloch sphere, respectively. This geometric interpretation provides an intuitive understanding of their action on qubit states. Two Pauli matrices commute if they are equal or if one of them is the identity matrix I; otherwise, they anticommute. This leads to the following relations:

$$ZX = -XZ = iY, \quad YZ = -ZY = iX, \quad XY = -YX = iZ. \tag{1.37}$$

In the computational basis, the Pauli gates act as follows. The X gate (also known as the NOT gate) acts as a bit flip:

$$X|x\rangle = |x \oplus 1\rangle, \tag{1.38}$$

where $x \in \{0, 1\}$ and $\oplus$ denotes addition modulo 2. The Z gate introduces as a phase flip:

$$Z|x\rangle = (-1)^x |x\rangle. \tag{1.39}$$

The Y gate combines the effects of X and Z with an additional i phase:

$$Y|x\rangle = i(-1)^x |x \oplus 1\rangle. \tag{1.40}$$

The computational, diagonal, and circular basis are eigenbasis of the Z, X and Y Pauli matrices respectively. That is why the computational, diagonal, and circular bases are sometimes referred to as the Z, X, and Y bases, respectively.

The Pauli group. All tensor products of n Pauli matrices, together with an overall phase of ± 1 or $\pm i$, generate the Pauli group $\mathcal{P}_n$:

$$\mathcal{P}_n = \{i^k P_1 \otimes \cdots \otimes P_n : k \in [0..3], P_i \in \{I, X, Y, Z\}\}. \tag{1.41}$$

We define the subset $\mathcal{P}_n^* \subset \mathcal{P}_n$ as the set of Pauli operators which have an overall phase of ± 1. We use the following notations and properties when working with Pauli operators:

- P_i denotes the ith Pauli matrix of a Pauli operator P. For instance, if $P = Z \otimes X$, then $P_1 = Z$ and $P_2 = X$.
- The term Pauli product designates a Pauli operator deprived of its sign.
- A Pauli operator P is said to be diagonal if and only if $P_i \in \{I, Z\}$ for all i. This is because both I and Z are diagonal matrices, so their tensor product will also be diagonal.
- The weight of a Pauli operator P, denoted $|P|$, is the number of non-identity Pauli matrices in its tensor product representation.

The commutation relations of Pauli operators are determined by their component-wise interactions. Two Pauli operators P and P' commute if there is an even number of indices i such that P_i anticommutes with P'_i. Otherwise, they anticommute.

1.2.1.2 Clifford gates

Clifford gates form a crucial class of quantum operations that play a fundamental role in various aspects of quantum computing, particularly in quantum error correction.

Single-qubit Clifford gates. While Pauli gates correspond to π rotations around the x-, y-, or z-axis of the Bloch sphere, single-qubit Clifford gates correspond to rotations by multiples of $\pi/2$ around these axes. For example, the S gate, also referred to as the phase gate, corresponds to a $\pi/2$ rotation around the z-axis of the Bloch sphere. It is defined by the following matrix:

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \tag{1.42}$$

Similarly, the $\sqrt{X}$ gate corresponds to a $\pi/2$ rotation around the x-axis. It is defined as:

$$\sqrt{X} = \frac{1}{2} \begin{pmatrix} 1+i & 1-i \\ 1-i & 1+i \end{pmatrix}. \tag{1.43}$$

All single-qubit Clifford gates can be generated using compositions of the S and $\sqrt{X}$ gates, up to a global phase. That is because these two gates generate $\pi/2$ rotations around two different axes the Bloch sphere. For example, the Hadamard gate, denoted by H, is defined as:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{1.44}$$

The Hadamard gate can be decomposed using S and $\sqrt{X}$, up to a global phase of $e^{i\pi/4}$, as follows:

$$H = e^{i\pi/4} S \sqrt{X} S \tag{1.45}$$

Global phases do not affect measurement outcomes in quantum mechanics, so this equivalence is sufficient for all practical purposes.

Two-qubit Clifford gates. Two-qubit Clifford gates are enabling the creation of entanglement, a key aspect of quantum computing. One of the most important two-qubit Clifford gates is the CNOT (Controlled-NOT) gate, defined by the following matrix:

$$\mathrm{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{1.46}$$

In the computational basis, the CNOT gate performs a conditional bit flip operation:

$$\mathrm{CNOT}\,|a\rangle\,|b\rangle = |a\rangle\,|a \oplus b\rangle \tag{1.47}$$

where $a, b \in \{0, 1\}$ and $\oplus$ denotes addition modulo 2. When we need to explicitly indicate which qubits the CNOT acts on, we use the notation $\mathrm{CNOT}_{A,B}$ where A denotes the control qubit and B the target qubit:

$$\mathrm{CNOT}_{A,B}\,|a\rangle_A\,|b\rangle_B = |a\rangle_A\,|a \oplus b\rangle_B\,. \tag{1.48}$$

Another useful two-qubit Clifford gate is the CZ (Controlled-Z) gate, defined as:

$$\mathrm{CZ} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}. \tag{1.49}$$

In the computational basis, the CZ gate performs a conditional phase flip operation:

$$\mathrm{CZ}\,|a\rangle\,|b\rangle = (-1)^{ab}\,|a\rangle\,|b\rangle\,. \tag{1.50}$$

Interestingly, the CZ gate can be constructed by conjugating a CNOT gate with Hadamard gates on the target qubit:

$$\mathrm{CZ} = (I \otimes H) \cdot \mathrm{CNOT} \cdot (I \otimes H) \tag{1.51}$$

This relationship arises because the Hadamard gate maps the computational basis to the diagonal basis, effectively transforming the conditional bit flip operation into a conditional phase flip operation.

The SWAP gate is defined by the following matrix:

$$\mathrm{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1.52}$$

In the computational basis, the SWAP gate operates as follows, by swapping the states of two qubits:

$$\mathrm{SWAP}\,|a\rangle\,|b\rangle = |b\rangle\,|a\rangle\,. \tag{1.53}$$

This gate can be decomposed into three CNOT gates:

$$\mathrm{SWAP} = \mathrm{CNOT}_{A,B} \cdot \mathrm{CNOT}_{B,A} \cdot \mathrm{CNOT}_{A,B}. \tag{1.54}$$

The Clifford group. The Clifford group $\mathcal{C}_n$ is defined as the set of unitary operators that normalize the Pauli group $\mathcal{P}_n$:

$$\mathcal{C}_n = \{U \mid U^\dagger P U \in \mathcal{P}_n, \forall P \in \mathcal{P}_n\}. \tag{1.55}$$

In other words, Clifford operators are mapping Pauli operators to Pauli operators under conjugation. An important characteristic of the Clifford group is its transitive action on the non-identity Pauli operators. As such, for each pair of Pauli operators $P, P' \in \mathcal{P}_n \setminus \{I^{\otimes n}\}$, there exists a Clifford operator $U \in \mathcal{C}_n$ such that $P' = U^\dagger P U$. The Clifford group can be generated by the set of gates $\{\text{CNOT}, S, H\}$, meaning that any Clifford operation can be decomposed into a sequence of these gates.

The tableau representation. The action of any n-qubit Clifford operation can be fully characterized by its effect on $2n$ Pauli operators belonging to $\mathcal{P}_n^*$, the subset of the Pauli group with phases ± 1 [17]. This compact representation forms the basis of the tableau representation. The $2n$ Pauli operators are divided into two sets:

- **Stabilizer generators:** n operators representing how the Clifford operation transforms the Pauli Z operators.
- **Destabilizer generators:** n operators representing how the Clifford operation transforms the Pauli X operators.

These $2n$ Pauli operators are efficiently encoded in a binary matrix of size $(2n+1) \times 2n$ called a tableau. For each Pauli operator, two bits are used for each qubit to encode the associated Pauli matrix (for example: 00 for I, 01 for X, 10 for Z, and 11 for Y), and one bit is used for its sign (0 for $+1$, 1 for -1).

The power of the tableau representation lies in its efficiency. It represents the action of a Clifford operation on all 4^n Pauli operators using only $O(n^2)$ bits. When a single-qubit or two-qubit Clifford gate is applied, the tableau can be updated in $O(n)$ time. This is because Clifford gates map Pauli operators to Pauli operators, which can be tracked by simple matrix operations on the tableau. In addition, when a Pauli basis measurement is applied, the tableau can also be updated in polynomial time. The outcome probabilities and post-measurement states can be computed directly from the tableau representation.

The tableau representation provides a constructive proof of the Gottesman-Knill theorem, which states that any quantum circuit composed solely of Clifford gates and Pauli measurements can be efficiently simulated on a classical computer [18]. This result enables efficient classical simulation of important quantum protocols, such as many quantum error correction schemes.

1.2.1.3 Universal gate sets

A set of quantum gates is called universal if any unitary operation can be implemented using only gates from this set. And it is approximately universal if any unitary operation can be approximated to arbitrary precision using only gates from this set. The Solovay-Kitaev theorem [19, 20] provides a constructive method for approximating any unitary operation efficiently using an approximately universal gate set. Specifically, it states that a unitary operation can be approximated within an error ϵ using $O(\log^c(1/\epsilon))$ gates from an approximately universal gate set, where $c < 4$. This theorem guarantees that different approximately universal gate sets are very similar in terms computational efficiency, as any gate sequence from one set can be efficiently translated into a sequence from another set while maintaining arbitrary precision.

The Clifford gates alone do not form an approximately universal gate set. However, the Clifford group plus any non-Clifford gate yields an approximately universal gate set. The T gate is a common choice for this non-Clifford gate, defined as:

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}. \tag{1.56}$$

This gate performs a rotation of $\pi/4$ around the Z axis in the Bloch sphere representation. An interesting property of the T gate is that $T^2 = S$, where S is the Clifford phase gate. The Clifford+T gate set is one of the most widely used approximately universal gate set.

Other common non-Clifford gates are the Toffoli (or Controlled-CNOT) gate and the CCZ (or Controlled-CZ) gate. These three-qubit gates are defined as follows:

$$\mathrm{Tof} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}, \quad \mathrm{CCZ} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix}. \tag{1.57}$$

In the computational basis, the Toffoli gate performs a conditional bit flip operation, extending the CNOT gate by adding a second control qubit:

$$\mathrm{Tof}\,|a\rangle\,|b\rangle\,|c\rangle = |a\rangle\,|b\rangle\,|c \oplus ab\rangle \tag{1.58}$$

Similarly, the CCZ gate performs a conditional phase flip operation, extending the CZ gate with an additional control:

$$\mathrm{CCZ}\,|a\rangle\,|b\rangle\,|c\rangle = (-1)^{abc}\,|a\rangle\,|b\rangle\,|c\rangle \tag{1.59}$$

Just as the CNOT and CZ gates are related by Hadamard conjugation, the Toffoli and CCZ gates share a similar relationship:

$$\mathrm{CCZ} = (I \otimes I \otimes H) \cdot \mathrm{Tof} \cdot (I \otimes I \otimes H). \tag{1.60}$$

The Toffoli and CCZ gates can be exactly implemented using the Clifford+T gate set. The minimal number of T gates required to implement them with this gate set is 7 [21]. However, by incorporating an additional qubit and a measurement, this T-count can be reduced to 4 [22].

1.2.2 Quantum circuits

Quantum circuits are the quantum analog of classical logic circuits. They provide a visual representation of quantum computations and are composed of quantum gates and measurements acting on qubits.

1.2.2.1 Quantum circuit diagrams

In a quantum circuit, qubits are represented by horizontal lines, and time progresses from left to right. Single-qubit gates are represented by boxes containing the gate symbol. For example, the H, S and T gates are represented as follows:

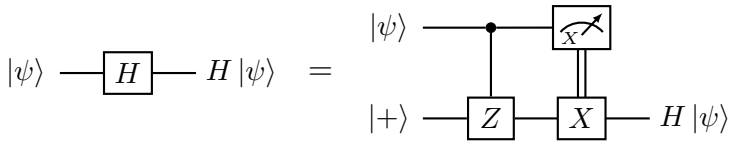


Figure 1.2: Hadamard gate gadgetization circuit.

Multi-qubit gates are typically represented by a vertical line connecting the affected qubits, with a solid black dot (•) on the control qubits. For example, the two-qubit CNOT and CZ gates are depicted as follows:

The three-qubit Toffoli and CCZ gates are represented in a similar way with an additional control qubit:

Measurements are depicted by a meter symbol with an X or Z label, depending on whether it is an X-basis measurement or Z-basis measurement:

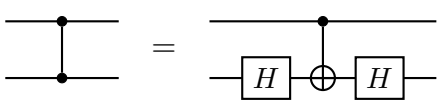
The X-basis measurement projects the qubit onto the eigenstates of the Pauli X operator, which are the diagonal basis states ($|+\rangle$ and $|-\rangle$), while the Z-basis measurement projects onto the computational basis states ($|0\rangle$ and $|1\rangle$).

Gates classically controlled by measurement outcomes are depicted with double lines connecting the measurement to the controlled operation. For instance, the circuit represented in Figure 1.2 performs an X-basis measurement with a classically controlled Pauli X gate depending on the measurement outcome.

1.2.2.2 Circuit identities

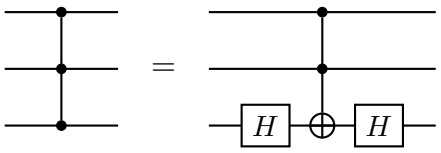
Circuit identities are fundamental relationships between quantum gates or sequences of gates that allow for the simplification, optimization, or transformation of quantum circuits.

For example, the relationship between the CNOT and CZ gates, as shown in Equation 1.51, can be represented by the following circuit equality:



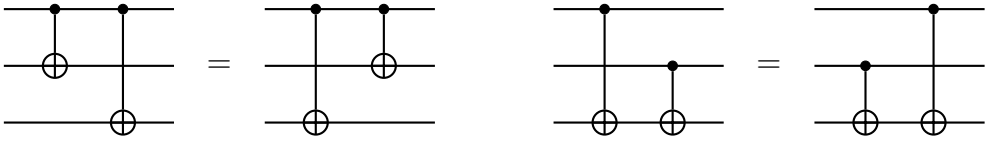
(1.61)

Analogously, the relationship between the CCZ and Toffoli gates, as shown in Equation 1.60, can be expressed by the following circuit equality:



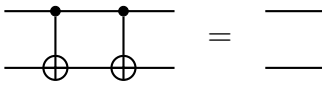
(1.62)

Some common circuit identities include gate commutation rules, which describe how gates can be reordered without affecting the overall computation. For instance, certain gates commute when applied to different qubits, allowing for parallelization or reordering to reduce circuit depth. Similarly, CNOT gates with the same target or control qubit commute:



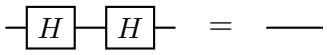
(1.63)

Other useful identities involve gate cancellation, where sequences of gates can be removed from the circuit. For example, applying two CNOT gates on the same qubits results in the identity operation, effectively canceling out both gates:



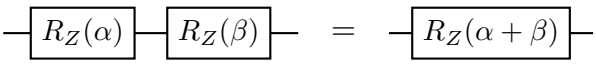
(1.64)

This property holds for all self-inverse quantum gates, such as the Hadamard gate:



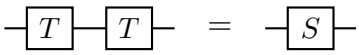
(1.65)

Some identities allow for gate simplification. For instance, two adjacent R_Z gates can be merged into a single R_Z gate:



(1.66)

When $\alpha = \beta = \pi/4$, this equality corresponds to the following simplification, which reduces the number of T gates in the circuit:



(1.67)

There exist numerous other circuit identities that are useful in various contexts. A particularly important identity, frequently used in this work, is the Hadamard gate gadgetization circuit, shown in Figure 1.2. This circuit implements the Hadamard operator without directly using the Hadamard gate by relying on an ancillary qubit prepared in the $|+\rangle$ state.

1.2.2.3 Circuit metrics

To evaluate and optimize quantum circuits, several key metrics are commonly used. The circuit depth corresponds to the length of the longest path from input to output in the circuit, where gates operating in parallel count as a single time step. This metric is crucial as it relates to the execution time of the quantum algorithm.

The circuit size or gate count is the total number of gates in the circuit. We also often refer to the number of specific gates. For example, the number of T gates, called the T-count, is a

critical metric in fault-tolerant quantum computing due to the high cost of implementing T gates in many error-correcting codes.

Other important metrics are the number of qubits and the number of ancillary qubits. Ancillary qubits are often needed for temporary storage or to implement certain operations more efficiently. For instance, the Hadamard gate gadgetization circuit shown in Figure 1.2 reduces the number of Hadamard gates by one (provided we can initialize a qubit in the $|+\rangle$ state and perform an X-basis measurement without the Hadamard gate) at the cost of using an ancillary qubit.

1.2.3 Global gates

The quantum gates discussed in Section 1.2.1 are all operating on a small, constant number of qubits, typically one or two. These gates serve as elementary building blocks in quantum circuits, from which any quantum operation can be constructed. Their fundamental nature stems from both theoretical considerations, as they can approximate any unitary operation to arbitrary precision, and practical implementations, as they are directly supported by most quantum computing architectures.

However, certain quantum architectures and error-correcting codes provide efficient fault-tolerant implementations of quantum gates operating on a larger number of qubits, typically scaling with the size of the quantum circuit. For instance, the surface code supports the implementation of the fan-out (or multi-target CNOT) gate and of multi-qubit Pauli rotations (when provided with appropriate magic states) [23, 24]. As such, these global gates represent an important aspect of quantum circuit design. Moreover, a fundamental challenge in quantum circuit design lies in efficiently decomposing these global operations into sequences of non-global gates when direct implementation is not available.

In this section, we present various global gates that are particularly relevant to the work presented in this manuscript. We also introduce some quantum operators that can be defined using these global gates, such as sequences of Pauli rotations and phase polynomials.

1.2.3.1 Multi-controlled NOT (MCX) gate

The multi-controlled NOT gate, denoted by MCX_n, acts on a $(n+1)$-dimensional computational basis vector $|\boldsymbol{x}\rangle$ as follows:

$$\mathrm{MCX}_n |\boldsymbol{x}\rangle = \left(\bigotimes_{i=0}^{n-1} |x_i\rangle \right) |x_n \oplus \prod_{i=0}^{n-1} x_i\rangle. \tag{1.68}$$

Note that we have $\mathrm{MCX}_1 = \mathrm{CNOT}$ and $\mathrm{MCX}_2 = \mathrm{Toffoli}$.

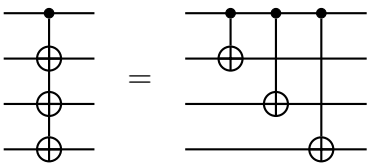
It has been shown in Reference [25] that an MCX_n gate can be implemented over the $\{\mathrm{Toffoli}, X\}$ gate set with a depth of $\mathcal{O}(\log_2(n))$ and a gate count of $\mathcal{O}(n)$, using one ancillary qubit. In addition, it has also been shown in Reference [26] that an MCX_n gate can be implemented over the $\{\mathrm{Toffoli}, X\}$ gate set with the same depth and gate count complexities of $\mathcal{O}(\log_2(n))$ and $\mathcal{O}(n)$ respectively, by using two dirty ancillary qubits (i.e. ancillary qubits in arbitrary states).

1.2.3.2 Fan-out gate

The fan-out operator, denoted by $\mathcal{F}_1^{(n)}$, acts on a $(n+1)$-dimensional computational basis vector $|\boldsymbol{x}\rangle$ as follows:

$$\mathcal{F}_1^{(n)} |\boldsymbol{x}\rangle = |x_0\rangle \bigotimes_{i=1}^{n} |x_i \oplus x_0\rangle \,. \tag{1.69}$$

It has been proven that this operator can be implemented with a CNOT-depth of $\mathcal{O}(\log_2(n))$ and a CNOT-count of $\mathcal{O}(n)$ [27, 28]. The operator $\mathcal{F}_1^{(n)}$ can be implemented by a multi-target CNOT gate with n targets, or a sequence of CNOT gates all sharing the same controlled qubit. For example, the following circuits are implementing the $\mathcal{F}_1^{(3)}$ operator:

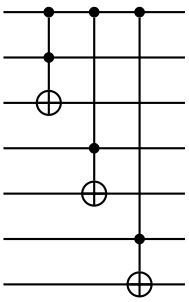


(1.70)

A natural generalization of this operator is the second-order fan-out operator $\mathcal{F}_2^{(n)}$, which replaces the target X gates with controlled-NOT gates. This operator acts on $(2n+1)$ qubits and transforms a computational basis state $|\boldsymbol{x}\rangle$ as follows:

$$\mathcal{F}_2^{(n)} |\boldsymbol{x}\rangle = |x_0\rangle \bigotimes_{i=1}^{n} |x_{2i-1}\rangle \, |x_{2i} \oplus x_0 x_{2i-1}\rangle \,. \tag{1.71}$$

For example, the following circuit is implementing the $\mathcal{F}_2^{(3)}$ operator:



(1.72)

We prove in Chapter 4 that this operator can be implemented using Toffoli and CNOT gates with a depth of $\mathcal{O}(\log_2(n))$ and a gate count of $\mathcal{O}(n)$.

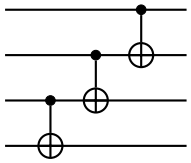
1.2.3.3 Ladder gate

The ladder operator, denoted by $\mathcal{L}_1^{(n)}$, acts on a $(n+1)$-dimensional computational basis vector $|\boldsymbol{x}\rangle$ as follows:

$$\mathcal{L}_1^{(n)} |\boldsymbol{x}\rangle = |x_0\rangle \bigotimes_{i=1}^{n} |x_i \oplus x_{i-1}\rangle \,. \tag{1.73}$$

Each qubit (except the first) is XORed with its predecessor in the sequence, which creates a ladder pattern of dependencies. For example, the $\mathcal{L}_1^{(3)}$ operator can be implemented with a

ladder of CNOT gates as follows:



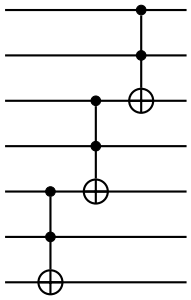
(1.74)

The ladder operator plays a crucial role in quantum ripple-carry adder implementations [29]. We prove in Chapter 4 that this operator can be implemented with a CNOT-depth of $\mathcal{O}(\log_2(n))$ and a CNOT-count of $\mathcal{O}(n)$. This implementation is used in Chapters 4 and 5 to improve the depth of some arithmetic quantum circuits.

A natural generalization of the $\mathcal{L}_1^{(n)}$ operator consists in replacing the ladder of CNOT gates by a ladder of Toffoli gates. This second-order ladder operator, denoted by $\mathcal{L}_2^{(n)}$, acts on a $(2n+1)$-dimensional computational basis vector $|\boldsymbol{x}\rangle$ as follows:

$$\mathcal{L}_2^{(n)} |\boldsymbol{x}\rangle = |x_0\rangle \bigotimes_{i=1}^{n} |x_{2i-1}\rangle \, |x_{2i} \oplus x_{2i-2}x_{2i-1}\rangle \, . \tag{1.75}$$

For example, the $\mathcal{L}_2^{(3)}$ operator can be implemented with a ladder of Toffoli gates as follows:



(1.76)

This second-order ladder operator also plays a crucial role in quantum ripple-carry adder implementations [29]. We prove in Chapter 4 that this operator can be implemented using gates with an MCX-depth of $\mathcal{O}(\log_2(n))$ and a MCX-count of $\mathcal{O}(n)$. By using the implementation discussed in Section 1.2.3.1, this leads to an implementation of the $\mathcal{L}_2^{(n)}$ operator over the $\{\text{Toffoli}, X\}$ gate set with a depth of $\mathcal{O}(\log_2(n))$ and a gate count of $\mathcal{O}(n \log_2(n))$. We use this result in Chapter 4 to present a polylogarithmic-depth quantum adder.

More generally, we can define $\mathcal{L}_{\boldsymbol{\alpha}}$ as an operator associated with a ladder composed of multiple MCX gates. Here, $\boldsymbol{\alpha}$ is a vector of integers that specifies the control and target qubits in the ladder. Specifically, the i-th MCX gate in the ladder has α_i as its target qubit and is controlled by all the qubits between α_{i-1} and α_i. For a vector $\boldsymbol{\alpha}$ of $k-1$ integer satisfying $0 \leq \alpha_0 < \alpha_1 < \ldots < \alpha_{k-2}$, the operator $\mathcal{L}_{\boldsymbol{\alpha}}$ acts on a $(\alpha_{k-2}+1)$-dimensional computational basis vector $|\boldsymbol{x}\rangle$ as follows:

$$\mathcal{L}_{\boldsymbol{\alpha}} |\boldsymbol{x}\rangle = \left(\bigotimes_{i=0}^{\alpha_0 - 1} |x_i\rangle \right) |x_{\alpha_0} \oplus \prod_{j=0}^{\alpha_0 - 1} x_j\rangle \bigotimes_{i=1}^{k-2} \left(\bigotimes_{j=\alpha_{i-1}+1}^{\alpha_i - 1} |x_j\rangle \right) |x_{\alpha_i} \oplus \prod_{j=\alpha_{i-1}}^{\alpha_i - 1} x_j\rangle. \tag{1.77}$$

Note that $\mathcal{L}_{\boldsymbol{\alpha}} = \mathcal{L}_1^{(n)}$ in the case where $\boldsymbol{\alpha} = (1, 2, 3, \ldots, n)$, and $\mathcal{L}_{\boldsymbol{\alpha}} = \mathcal{L}_2^{(n)}$ in the case where $\boldsymbol{\alpha} = (2, 4, 6, \ldots, 2n)$.

1.2.3.4 Pauli rotations

Given a Pauli operator $P \in \mathcal{P}_n^*$ and an angle $\theta \in \mathbb{R}$, a Pauli rotation $R_P(\theta)$ is defined as follows:

$$R_P(\theta) = \exp(-i\theta P/2) = \cos(\theta/2)I - i\sin(\theta/2)P. \tag{1.78}$$

For example, the T gate is a $\pi/4$ Pauli Z rotation:

$$T = R_Z(\pi/4). \tag{1.79}$$

In fact, any unitary gate can be represented as a product of Pauli rotations. For example, the CNOT, S and H gates can be defined as follows:

$$\begin{aligned} \text{CNOT} &= R_{ZX}(\pi/2)R_{ZI}(-\pi/2)R_{IX}(-\pi/2), \\ S &= R_Z(\pi/2), \\ H &= R_X(\pi)R_Y(\pi/2). \end{aligned}$$

A Pauli operator $P \in \mathcal{P}_n$ conjugated by a Clifford gate $U \in \mathcal{C}_n$ is always equal to another Pauli operator $P' \in \mathcal{P}_n$, i.e. $U^\dagger P U = P'$. This fact also holds when a Pauli rotation is conjugated by $U \in \mathcal{C}_n$:

$$U^\dagger R_P(\theta) U = R_{U^\dagger P U}(\theta) = R_{P'}(\theta). \tag{1.80}$$

That is why a unitary gate U, representing the operation performed by a quantum circuit acting on n qubits and composed of Clifford gates and single-qubit rotation gates, can always be described by a sequence of Pauli rotations and a final Clifford operator $C \in \mathcal{C}_n$ [21]:

$$U = e^{i\phi} C \left(\prod_{i=1}^{m} R_{P_i}(\theta_i) \right) \tag{1.81}$$

where m is the number of non-Clifford rotation gates in the circuit and P_i is a Pauli product of size n different from $I^{\otimes n}$, for all i. Two Pauli rotations $R_P(\theta)$ and $R_{P'}(\theta')$, where θ and θ' are satisfying $\theta \neq 0 \pmod{2\pi}$ and $\theta' \neq 0 \pmod{2\pi}$, commute if and only if P commutes with P'.

Chapter 2

Quantum Error Correction and Fault-Tolerant Quantum Computing

The development of quantum error correction and fault-tolerant protocols represents one of the most significant breakthroughs in quantum computing, offering a pathway to overcome the inherent fragility of quantum information. Quantum error correction addresses the fundamental challenge of protecting quantum information from errors, for example by encoding logical qubits across multiple physical qubits. It includes error detection and recovery procedures to maintain quantum state coherence far beyond what is achievable with individual physical qubits. Building upon these error correction techniques, fault-tolerant quantum computing enables operations between encoded quantum states while maintaining their protected status. The main property of fault tolerance is that it should remain effective even when all components of the quantum system are imperfect.

The first quantum error-correcting codes were proposed by Peter Shor [30] and Andrew Steane [31]. These early codes demonstrated that quantum information could be encoded across multiple physical qubits in such a way that some errors could be detected and corrected, laying the groundwork for more sophisticated quantum error-correcting codes. Daniel Gottesman introduced the stabilizer formalism in 1997 [32], providing a powerful framework for understanding and constructing many quantum error-correcting codes. The first scheme for fault-tolerant quantum computation was proposed by Peter Shor in 1996 [33]. This work was followed by the development of threshold theorems [34, 35], which established that a quantum computer could perform arbitrarily long computations as long as the error rates of its components are below a certain threshold.

Among the various approaches to quantum error correction, the surface code has emerged as one of the most promising candidates to achieve practical and reliable quantum computing. Introduced by Alexei Kitaev [36] and further developed by many others, the surface code offers several crucial advantages. It has a high error threshold, requires only local interactions, and the error detection measurements involve only small groups of nearby qubits. Additionally, the code distance can be increased by simply expanding the surface code lattice, allowing for scalable quantum error correction. Several methods have been proposed to achieve universal fault-tolerant quantum computation with the surface code. An effective approach relies on lattice surgery [37], which performs fault-tolerant operations by cutting and stitching the lattice structure of the surface code, and magic state distillation for implementing T gates [38].

This chapter provides an introduction to quantum error correction and fault-tolerant quantum computing by focusing on the surface code lattice surgery approach with magic state distillation.

2.1 Quantum error correction

2.1.1 Quantum error correction principles

Three key properties of quantum systems distinguish them from classical systems and fundamentally shape the approach to quantum error correction:

- **The no-cloning theorem:** Unlike classical information, quantum information cannot be copied. Specifically, it is impossible to create exact copies of arbitrary unknown quantum states. This theorem places fundamental constraints on how we can implement redundancy in quantum error correction.
- **The measurement problem:** Quantum measurements generally disturb or collapse quantum states, making it impossible to directly measure a quantum state without potentially destroying the information it carries. This presents a fundamental challenge for error detection, as we cannot simply read out the state to check for errors without disturbing the quantum computation.
- **Continuous nature of errors:** While classical errors are discrete (bit flips), quantum errors can be continuous, such as arbitrary rotations in the Bloch sphere.

To circumvent the no-cloning theorem while still using redundancy, quantum error correcting codes encode quantum information across multiple physical qubits to create protected logical qubits.

The measurement problem is addressed by performing specially designed multi-qubit measurements called syndrome measurements. These extract error information without disturbing the encoded quantum state in the logical qubits.

Finally, while quantum errors can be continuous, a remarkable feature of quantum error correction is its ability to discretize these errors through the projective nature of syndrome measurements. For example, an error involving a small rotation around the X axis can be expressed as a superposition of an identity operation (no error) and a bit-flip (X) error with certain probabilities. When syndrome measurements are performed, this small rotation error is projected onto one of these two possibilities. As a result of this discretization, errors in quantum systems are typically categorized into three fundamental types: bit-flip errors, phase-flip errors, or a combination of both. These error types correspond to the action of the Pauli operators X, Z and Y.

In addition to these errors, quantum systems are also susceptible to leakage errors, where a qubit leaves the computational subspace (typically the two-level subspace spanned by $|0\rangle$ and $|1\rangle$) and enters higher energy states outside this subspace. Leakage errors are particularly challenging because standard error correction codes, designed for errors within the computational basis, may fail to detect when a qubit has leaked to higher energy states. Specialized techniques or modifications to standard error correction protocols are often necessary to handle these errors effectively [39–41].

2.1.2 The stabilizer formalism

The stabilizer formalism, introduced by Daniel Gottesman [32], provides an efficient framework for describing a broad class of quantum error-correcting codes. This approach is fundamental to the study of quantum error correction because it enables the systematic construction and analysis of quantum codes using group theory and linear algebra. Since its inception, the stabilizer

formalism has become a central tool in the theory of quantum error correction, supporting the development of numerous practical quantum error-correcting codes.

2.1.2.1 Stabilizer states

A quantum state $|\psi\rangle$ is said to be stabilized by an operator S if it is an eigenstate of S with eigenvalue $+1$:

$$S|\psi\rangle = |\psi\rangle \tag{2.1}$$

In other words, the operator S leaves the state $|\psi\rangle$ unchanged. This simple property forms the foundation of the stabilizer formalism.

A stabilizer state is any quantum state that can be generated by applying Clifford operations to computational basis states. For example, the computational basis state $|0\rangle$ is stabilized by the Pauli operator Z:

$$Z|0\rangle = |0\rangle. \tag{2.2}$$

Similarly, the diagonal basis state $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ is stabilized by the Pauli operator X:

$$X|+\rangle = |+\rangle. \tag{2.3}$$

The concept extends naturally to multi-qubit systems. For instance, the two-qubit Bell state $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ is stabilized by both XX and ZZ:

$$XX|\Phi^+\rangle = |\Phi^+\rangle, \tag{2.4}$$
$$ZZ|\Phi^+\rangle = |\Phi^+\rangle. \tag{2.5}$$

2.1.2.2 Stabilizer groups

A stabilizer group, denoted $\mathcal{S}$, is an abelian subgroup of the Pauli group that contains all operators stabilizing a given quantum state. The abelian property (meaning all elements commute with each other) ensures that all operators in the group share a common eigenspace. A stabilizer group can be completely specified by its generators: a minimal set of independent operators that generate the full group through multiplication.

For example, consider again the Bell state $|\Phi^+\rangle$. Its complete stabilizer group is:

$$\mathcal{S} = \{II, XX, ZZ, -YY\} \tag{2.6}$$

This entire group can be generated by just two independent generators: $\{XX, ZZ\}$. The remaining elements can be obtained through multiplication of these generators.

The number of independent generators needed to specify a stabilizer group for an n-qubit system cannot exceed n. This upper bound arises because each independent generator halves the dimension of the stabilized space, and an n-qubit system has dimension 2^n.

2.1.2.3 Stabilizer codes

The stabilizer formalism provides an elegant way to define quantum error-correcting codes through the specification of stabilizer operators rather than explicit codewords. A stabilizer code is defined as the joint $+1$ eigenspace of the elements in a stabilizer group $\mathcal{S}$:

$$\mathcal{C} = \{|\psi\rangle \mid S|\psi\rangle = |\psi\rangle \ \forall S \in \mathcal{S}\} \tag{2.7}$$

For example, consider the three-qubit bit flip code which encodes one logical qubit into three physical qubits to protect against bit flip errors. Rather than explicitly writing out the codewords:

$$|0_L\rangle = |000\rangle \tag{2.8}$$
$$|1_L\rangle = |111\rangle \tag{2.9}$$

we can specify the code through two stabilizer generators:

$$S_1 = Z_1 Z_2 \tag{2.10}$$
$$S_2 = Z_2 Z_3 \tag{2.11}$$

Any state $|\psi\rangle$ in the code space must satisfy:

$$S_1|\psi\rangle = |\psi\rangle \tag{2.12}$$
$$S_2|\psi\rangle = |\psi\rangle \tag{2.13}$$

The stabilizer representation provides direct insights into the code's properties. A particularly important relationship is that between the number of physical qubits n, the number of logical qubits k, and the number of independent stabilizer generators s:

$$k = n - s. \tag{2.14}$$

That is because each stabilizer generator constrains one degree of freedom, effectively halving the dimension of the allowed space. With s independent generators, the dimension of the code space becomes 2^{n-s}. Since this must equal 2^k where k is the number of logical qubits (as each logical qubit doubles the dimension of the code space), we obtain the equation $k = n - s$. For example, in the three-qubit bit flip code, we have $n = 3$ physical qubits and $s = 2$ stabilizer generators, yielding $k = 1$ logical qubit. This matches our understanding that the code encodes one logical qubit into three physical qubits.

2.1.2.4 Error detection mechanism

The error detection mechanism in stabilizer codes is based on the commutation relations between the Pauli errors and the stabilizers. A Pauli error $E \in \mathcal{P}_n$ is detectable by a stabilizer code if it anticommutes with at least one of the stabilizer generators. For an encoded state $|\psi\rangle$, if E anticommutes with a stabilizer generator S, the measurement outcome of S will be -1, indicating the presence of an error:

$$SE|\psi\rangle = -ES|\psi\rangle = -E|\psi\rangle. \tag{2.15}$$

In the case where E commutes with all the stabilizer generators, then E does not change the measurement outcomes of the stabilizers. This implies that E either leaves the encoded state unchanged and is therefore a stabilizer, or it is an error that the code fails to detect.

The error syndrome is the set of outcomes obtained from measuring all stabilizer generators. By analyzing the error syndrome, one can infer the most likely error that occurred and apply the appropriate correction. This process is referred to as decoding and should be fast and efficient for the code to be effective.

The code distance determines the maximum number of errors that the code can reliably correct. It is defined as the weight of the smallest Pauli error $E \in \mathcal{P}_n$ that commutes with all stabilizer generators but is not itself a stabilizer. The weight of a Pauli operator is the number

of qubits on which it acts nontrivially. For example, a code with a distance d can correct up to $\left\lfloor \frac{d-1}{2} \right\rfloor$ errors using a perfect decoder. If an error of weight greater than $\left\lfloor \frac{d-1}{2} \right\rfloor$ occurs, the code may not be able to correct it, leading to a potential decoding failure. A stabilizer code which uses n physical qubits to encode k logical qubits with a code distance of d is denoted by $[[n, k, d]]$.

2.1.2.5 Logical gates

Logical gates in stabilizer codes are operators that act on the encoded states while preserving the codespace. These operators allow us to perform quantum computations directly on the encoded logical qubits. A logical gate L of a stabilizer code $\mathcal{C}$ is a unitary operator that maps the codespace to itself:

$$L|\psi\rangle \in \mathcal{C} \quad \text{for all} \quad |\psi\rangle \in \mathcal{C}. \tag{2.16}$$

A key characteristic of logical operators is their interaction with the stabilizer group. For a stabilizer code $\mathcal{C}$, a unitary operator L is a logical operator if and only if it maps elements of the stabilizer group $\mathcal{S}$ to stabilizers:

$$L^\dagger S L|\psi\rangle = |\psi\rangle, \quad \forall S \in \mathcal{S}, \quad \forall |\psi\rangle \in \mathcal{C}. \tag{2.17}$$

Logical operators can be classified into two categories. Trivial logical operators are the stabilizers of the codespace. They do not affect the encoded states as they are acting as the identity operation on the logical qubits. Non-trivial logical operators are those which are not stabilizers of the codespace, effectively performing non-identity operations on the logical qubits.

The fundamental logical operators for any stabilizer code are the Pauli logical operators $\bar{X}$ and $\bar{Z}$, corresponding to the Pauli gates X and Z respectively. A key property of Pauli logical operators is that they must necessarily commute with the elements of the stabilizer group $\mathcal{S}$. If a Pauli logical operator L were to anticommute with any stabilizer $S \in \mathcal{S}$ for a stabilizer code $\mathcal{C}$, then, because every Pauli operator either commutes or anticommutes, we would have

$$L^\dagger S L\,|\psi\rangle = -S\,|\psi\rangle = -\,|\psi\rangle\,, \quad \forall\,|\psi\rangle \in \mathcal{C}. \tag{2.18}$$

which contradicts the definition of a logical operator. Also, the Pauli logical operators $\bar{X}$ and $\bar{Z}$ must not be a product of the stabilizers as they must act non-trivially on the codespace. Additionally, for the logical operators $\bar{X}$ and $\bar{Z}$ to faithfully represent the logical gates X and Z respectively, they must also anticommute:

$$\bar{X}\bar{Z} = -\bar{Z}\bar{X}. \tag{2.19}$$

Because Clifford operators are mapping Pauli operators to Pauli operators, their action can be completely characterized by how they transform the Pauli logical operators. For example, the logical Hadamard gate $\bar{H}$ transforms the logical computational basis states $|0\rangle_L$ and $|1\rangle_L$ to the logical diagonal basis states $|+\rangle_L$ and $|-\rangle_L$, and vice versa. Equivalently, $\bar{H}$ is the logical Hadamard operator if and only if it maps the logical operator $\bar{X}$ to the logical operator $\bar{Z}$, and vice versa:

$$\bar{H}\bar{X}\bar{H}^\dagger = \bar{Z}, \tag{2.20}$$

$$\bar{H}\bar{Z}\bar{H}^\dagger = \bar{X}. \tag{2.21}$$

Logical operators can often be implemented transversally, meaning that the logical gate is realized by applying identical single-qubit operations to each physical qubit independently.

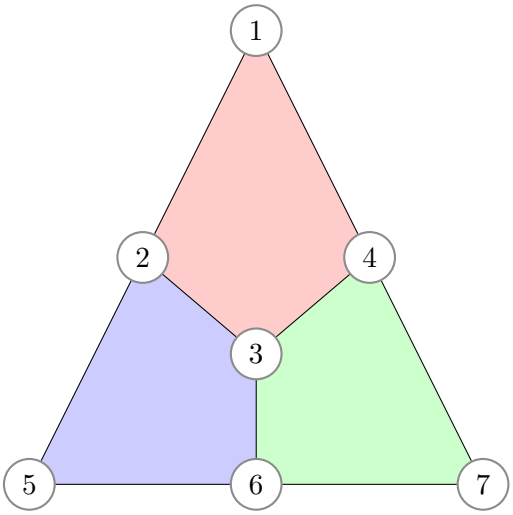


Figure 2.1: Illustration of the $[[7, 1, 3]]$ Steane code. Each vertex represents a physical qubit labeled from 1 to 7. The three colored regions represent the stabilizer generators: the red region (vertices $\{1, 2, 3, 4\}$) corresponds to stabilizers $S_X^{(1)}$ and $S_Z^{(1)}$, the blue region (vertices $\{2, 3, 5, 6\}$) corresponds to $S_X^{(2)}$ and $S_Z^{(2)}$, and the green region (vertices $\{3, 4, 6, 7\}$) corresponds to $S_X^{(3)}$ and $S_Z^{(3)}$.

Transversal gates are particularly desirable as they prevent the propagation of errors between different physical qubits. However, as stated by the Eastin-Knill theorem [42], no quantum error-correcting code can implement an approximately universal set of logical gates transversally. Some complementary methods to achieve universal fault-tolerant quantum computation are discussed in Section 2.2.

2.1.2.6 Example: the Steane code

The Steane code, introduced by Andrew Steane in 1996 [31], is a prominent example of a stabilizer code. As a $[[7, 1, 3]]$ stabilizer code, it encodes one logical qubit into seven physical qubits with a distance of 3, allowing it to correct any single-qubit error.

The stabilizer group of the Steane code is generated by six stabilizers, divided into three X-type and three Z-type operators:

$$S_X^{(1)} = XXXXIII \qquad S_Z^{(1)} = ZZZZIII \tag{2.22}$$

$$S_X^{(2)} = IXXIXXI \qquad S_Z^{(2)} = IZZIZZI \tag{2.23}$$

$$S_X^{(3)} = IIXXIXX \qquad S_Z^{(3)} = IIZZIZZ \tag{2.24}$$

These stabilizer generators are illustrated in Figure 2.1.

The Pauli logical operators $\bar{X}$ and $\bar{Z}$ of the Steane code can be defined as:

$$\bar{X} = X^{\otimes 7}, \tag{2.25}$$

$$\bar{Z} = Z^{\otimes 7}. \tag{2.26}$$

We can easily verify that these operators commute with all the stabilizer generators, and they are not products of the stabilizers, which implies that they are non-trivial Pauli operators. They

also anticommute with each other, which means that they implement different Pauli operators. Note that these are not the smallest Pauli logical operators. The smallest Pauli logical operators are of size 3, which is why the code has a distance of 3. For example, by multiplying $\bar{X}$ and $\bar{Z}$ by the stabilizer generators $S_X^{(1)}$ and $S_Z^{(1)}$ respectively, we get the following Pauli logical operators:

$$\bar{X}' = \bar{X}S_X^{(1)} = IIIIXXX, \tag{2.27}$$

$$\bar{Z}' = \bar{Z}S_Z^{(1)} = IIIIZZZ. \tag{2.28}$$

Each single-qubit Pauli error produces a unique syndrome, enabling error correction. For instance, consider a syndrome with measurement outcomes of -1 for the stabilizers $S_Z^{(1)}$ and $S_Z^{(2)}$, and $+1$ for all other stabilizer generators. This syndrome indicates a Pauli X error on the second qubit, as it is the only single-qubit error that anticommutes with $S_Z^{(1)}$ and $S_Z^{(2)}$ while commuting with all other stabilizers.

The Steane code can implement the complete Clifford group using transversal logical gates. For example, the logical Hadamard gate $\bar{H}$ is implemented by applying physical Hadamard gates to each qubit:

$$\bar{H} = H^{\otimes 7}. \tag{2.29}$$

To verify that $\bar{H}$ is indeed a logical gate, we need to ensure it maps each stabilizer generator to a stabilizer. This can easily be confirmed by noticing that $\bar{H}$ transforms the X-type stabilizers to the Z-type stabilizers and vice versa, preserving the stabilizer group:

$$\bar{H}^\dagger S_X^{(i)} \bar{H} = S_Z^{(i)}, \quad \forall i \in \{1,2,3\}, \tag{2.30}$$

$$\bar{H}^\dagger S_Z^{(i)} \bar{H} = S_X^{(i)}, \quad \forall i \in \{1,2,3\}. \tag{2.31}$$

Moreover, we can verify that $\bar{H}$ correctly implements the logical Hadamard gate by checking its effect on the logical Pauli operators. Specifically, $\bar{H}$ should map $\bar{X}$ to $\bar{Z}$ and $\bar{Z}$ to $\bar{X}$. We have

$$\bar{H}\bar{X}\bar{H}^\dagger = H^{\otimes 7}X^{\otimes 7}H^{\otimes 7} = Z^{\otimes 7} = \bar{Z}, \tag{2.32}$$

$$\bar{H}\bar{Z}\bar{H}^\dagger = H^{\otimes 7}Z^{\otimes 7}H^{\otimes 7} = X^{\otimes 7} = \bar{X}. \tag{2.33}$$

This transformation is consistent with the action of the Hadamard gate on the Pauli operators, confirming that $\bar{H}$ is acting as a logical Hadamard gate for the Steane code.

2.1.3 The surface code

The surface code is a popular stabilizer code which offers several crucial advantages such as a high error threshold, local interactions, and low-weight stabilizer measurements. A high error threshold means that the surface code tolerates a relatively large number of errors before failure, making it more robust compared to many other quantum error-correcting codes. Local interactions refer to the fact that the operations required to implement the code involve only nearby qubits. This simplifies the physical implementation of the code, as it reduces the complexity of wiring and minimizes the need for long-range interactions, which can be challenging to achieve with high fidelity. Lastly, low-weight stabilizer measurements mean that error detection involves only a small number of qubits at a time. This not only improves measurement efficiency but also minimizes the risk of errors propagating across the system

This stabilizer code family has several variants, including the original toric code introduced by Kitaev [36], the planar surface code [43,44], and the rotated surface code [45]. While the toric

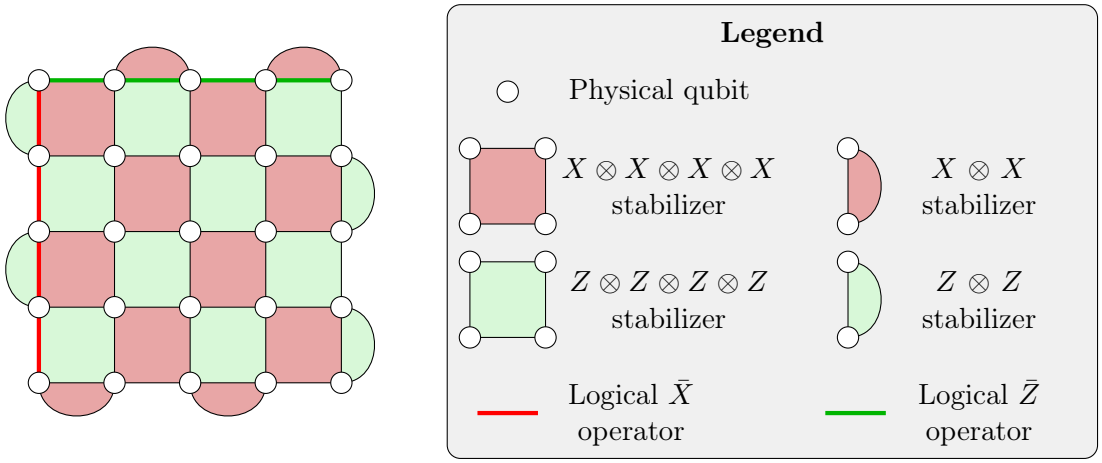


Figure 2.2: Surface code layout for a single logical qubit of size 5×5. The lattice consists of physical qubits represented by vertices connected by stabilizer measurements. Red regions indicate X-type stabilizer measurements, while green regions represent Z-type stabilizer measurements.

code is defined on a torus, the planar surface code operates on a flat lattice with boundaries, making it more practical for physical implementation. The rotated surface code offers improved resource efficiency by reducing the number of physical qubits required per logical qubit while maintaining the same code distance. The surface code presented here is the rotated surface code.

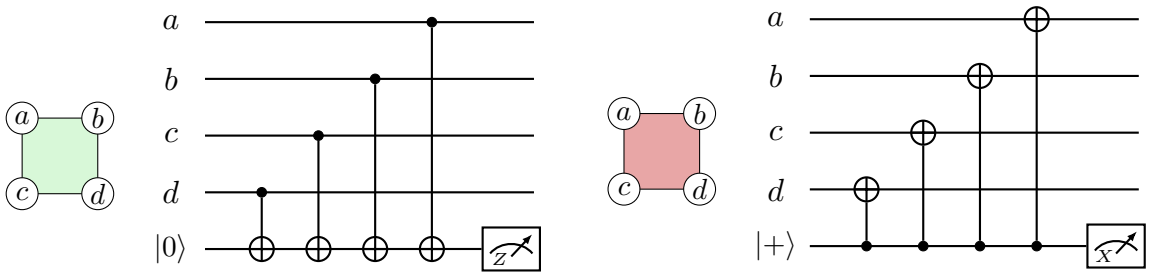
2.1.3.1 Stabilizers

The stabilizers of the surface code can be arranged on a two-dimensional grid where physical qubits involved in the measurement stabilizers are placed at the vertices. The code is composed of two types of stabilizers: weight-4 stabilizers and weight-2 stabilizers at the boundaries.

The weight-4 stabilizers are arranged in a checkerboard pattern, alternating between X and Z stabilizers. This arrangement implies that each X stabilizer is surrounded by Z stabilizers, and vice versa. This pattern ensures proper commutation relations between all stabilizers, as no X and Z stabilizers have an odd number of shared qubits.

Weight-2 stabilizers are placed on the boundaries, which are of two types: smooth boundaries associated with weight-2 X stabilizers and rough boundaries associated with weight-2 Z stabilizers. To encode a single logical qubit, there must be two smooth and two rough boundaries. Weight-2 X stabilizers are positioned between pairs of qubits at the smooth boundaries that would have been involved in weight-4 Z stabilizers if the lattice continued. Analogously, weight-2 Z stabilizers are positioned between pairs of qubits at the rough boundaries that would have been involved in weight-4 X stabilizers if the lattice continued. A representation of the surface code stabilizers in a grid layout is provided in Figure 2.2. The quantum circuits for weight-4 stabilizer measurements are given in Figure 2.3. The circuits for weight-2 stabilizer measurements are similar but with two CNOT gates instead of four.

In total, for a lattice of size $n \times n$, the number of weight-4 stabilizers is $(n-1)^2$, and the number of weight-2 stabilizers at the boundaries is $2(n-1)$. As such, the total number of stabilizers is $(n-1)^2 + 2(n-1) = n^2 - 1$, which is equal to the number of physical qubits minus one and therefore satisfies the stabilizer condition to encode one logical qubit.



(a) $Z_aZ_bZ_cZ_d$ stabilizer measurement. (b) $X_aX_bX_cX_d$ stabilizer measurement.

Figure 2.3: Quantum circuits for stabilizer measurements in the surface code.

2.1.3.2 Logical operators

A sequence of k Pauli X gates $X_{\alpha_1}, \ldots, X_{\alpha_k}$ is defined as an X-chain if and only if for each consecutive pair of gates X_{α_i} and $X_{\alpha_{i+1}}$, there exists a Z-type stabilizer S that acts non-trivially on both qubits α_i and α_{i+1}, and such that S commutes with the Pauli operator formed by the entire sequence $X_{\alpha_1}, \ldots, X_{\alpha_k}$. Analogously, a sequence of k Pauli Z gates $Z_{\alpha_1}, \ldots, Z_{\alpha_k}$ is defined as a Z-chain if and only if for each consecutive pair of gates Z_{α_i} and $Z_{\alpha_{i+1}}$, there exists an X-type stabilizer S that acts non-trivially on both qubits α_i and α_{i+1}, and such that S commutes with the Pauli operator formed by the entire sequence $Z_{\alpha_1}, \ldots, Z_{\alpha_k}$.

If an X-chain commutes with all the stabilizers, then it is a Pauli logical operator by definition. Either it is a product of the stabilizers and therefore is a trivial Pauli logical operator. Or it is not a product of the stabilizers and is a non-trivial Pauli logical operator performing a Pauli operation on the encoded qubit. In such a case, the first and the last Pauli X gates of the chain must be acting on qubits located on smooth boundaries. Moreover, these two qubits must be on two different smooth boundaries. Indeed, in all other cases, the X-chain encloses a set of X stabilizers, and can therefore be expressed as a product of these stabilizers. That is why non-trivial Pauli logical operators $\bar{X}$ of a logical qubit encoded in the surface code are X-chains connecting the two smooth boundaries. Similarly, non-trivial Pauli logical operators $\bar{Z}$ of a logical qubit encoded in the surface code are Z-chains connecting the two rough boundaries. As these chains connect opposite boundaries, they must necessarily cross an odd number of times, which implies that $\bar{X}$ is anticommuting with $\bar{Z}$. Examples of Pauli logical $\bar{X}$ and $\bar{Z}$ operators are given in Figure 2.2.

2.1.3.3 Decoding

Consider a Pauli X error occurring on a qubit associated with a vertex of the lattice. This error will lead to measurement outcomes of -1 for the adjacent Z-type stabilizers involving that qubit, indicating the presence of an error. However, when multiple errors occur, the decoding problem becomes more challenging. For instance, an X-chain of errors will yield measurement outcomes of -1 only for the Z-type stabilizers located at the endpoints of the chain. An endpoint of an X-chain can even be undetected if it is connected to a smooth boundary, as each qubit in a smooth boundary is involved in only one Z-type stabilizer. Moreover, different X-chains can lead to identical syndrome measurements, making the identification of the actual error pattern non-trivial. To correct the Pauli X errors, the decoder must determine the most likely set of X-chains consistent with the observed Z-type stabilizers measurements. That is, if the outcome of a Z-type stabilizer measurement is -1, then the stabilizer must be at the endpoint of an

X-chain.

Finding the most likely set of X-chains consistent with the syndrome measurements requires minimizing the sum of the lengths of the X-chains, as error patterns involving more qubits are less likely to occur. This optimization problem can be solved using minimum-weight perfect matching algorithms. The algorithm treats the smooth boundaries and the locations of -1 syndrome measurements as vertices in a graph and finds the optimal pairing that minimizes the total distance between paired vertices. Each pair of vertices then defines an X-chain representing the most likely error path connecting them.

Note that the decoder does not need to find the exact set of errors that occurred. For example, two X-chains are equivalent if they differ by a product of X-type stabilizers. The decoder fails when the set of Pauli X errors plus the set of the Pauli X gates forming the decoded X-chains are not a product of stabilizers. In such a case, a Pauli logical $\bar{X}$ operator has been applied, leading to a logical error.

The decoding of Pauli Z errors follows the same principles as the decoding of Pauli X errors, but with the roles of X-type and Z-type stabilizers reversed. The same minimum-weight perfect matching algorithm can be applied independently two times to decode the X and the Z errors. Pauli Y errors are equivalent to an X error and a Z error and are therefore handled naturally by this decoding scheme. A decoding example of a surface code logical qubit is presented in Figure 2.4.

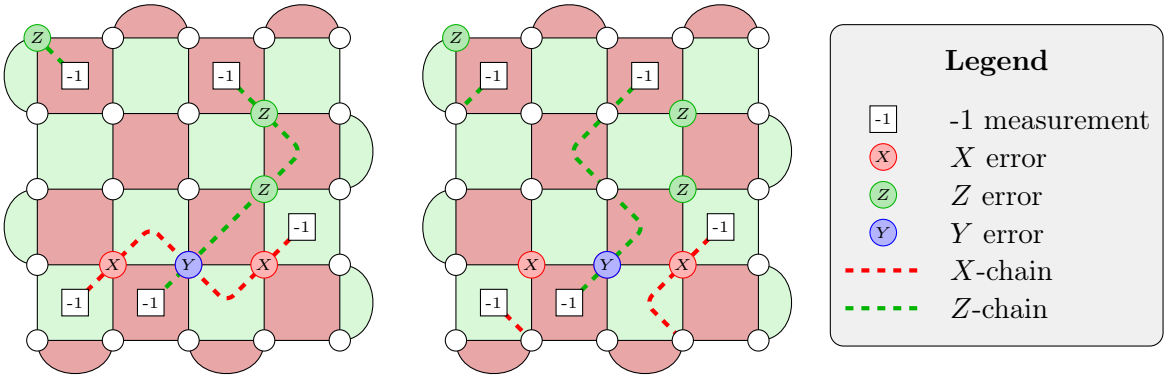
The minimum-weight perfect matching problem can be solved efficiently, for example by using Edmonds' blossom algorithm, which has a time complexity of $\mathcal{O}(n^3)$ where n is the number of vertices [46]. The time performance of a decoder is also of primordial importance in order for it to be used for real-time error correction in large-scale quantum devices. As such, many other decoding algorithms have been developed offering different trade-offs between time complexity and error-threshold [47–51].

The error threshold of a decoder refers to the maximum physical error rate below which increasing the code distance leads to better logical error rates. For the surface code with perfect measurements and optimal decoding, this threshold is approximately 11% for independent X and Z errors [52]. However, practical decoders operating with realistic noise models and measurement errors typically achieve lower thresholds. For instance, when considering measurement errors, additional rounds of syndrome measurements are required to achieve reliable error correction, and the threshold typically drops to around 1% for realistic noise models [53].

2.2 Fault-tolerant quantum computing

In order to perform reliable and universal quantum computation with the surface code, a crucial requirement is the fault-tolerant implementation of a universal gate set, such as the Clifford+T gate set.

Clifford gates can be implemented in the surface code through various fault-tolerant techniques. For instance, single-qubit Clifford gates can be realized using code deformation methods based on twists, a type of defect in the lattice [54–57]. While the CNOT gate can be realized transversally in the surface code, this approach presents significant practical challenges as it compromises the two-dimensional connectivity that makes the surface code particularly attractive for physical implementations. This has driven the development of alternative approaches for implementing the CNOT gate fault-tolerantly. Notable among these are braiding techniques [58, 59], which implement logical operations through topological manipulations of defects, and lattice surgery [37], which operates by merging and splitting surface code patches.



(a) (b)

Figure 2.4: Example of surface code error decoding on a distance-5 surface code logical qubit. Pauli X, Z, and Y errors are represented on the vertices (representing qubits) in red, green, and blue, respectively. Stabilizer measurements with outcomes -1 are depicted by a -1 box. The X-chains (red dashed lines) and Z-chains (green dashed lines) represent the solution produced by a minimum-weight perfect matching decoder. Subfigure 2.4(a) shows a decoding solution where the sum of the chain lengths is equal to 7. Subfigure 2.4(b) shows an equivalent successful decoding solution with different X-chains and Z-chains but with the same total weight of 7.

To achieve universality, the Clifford group must be supplemented with a non-Clifford gate, such as the T gate. The fault-tolerant implementation of the T gate can be accomplished through magic state distillation, a process that creates high-fidelity magic states from multiple noisy copies. A magic state is a quantum state that, when combined with Clifford operations, can be used to implement the T gate through gate teleportation protocols.

In this section, we present one of the most popular approaches for achieving fault-tolerant universal quantum computation with the surface code, which is based on lattice surgery and magic state distillation.

2.2.1 Lattice surgery

Lattice surgery is a method for performing fault-tolerant operations between multiple logical qubits encoded in the surface code. The logical operations are realized by manipulating the boundaries between surface code patches. The two fundamental operations in lattice surgery are the split and the merge operations. These operations come in two varieties: rough and smooth, depending on the type of boundaries on which they are performed.

2.2.1.1 Merge operation

The merge operation between two qubits can be visualized as merging the boundaries of two surface code patches, effectively combining the two logical qubits into a single logical qubit. This is achieved by activating new stabilizer measurements between the boundaries of the patches. In the following, we will consider two single-qubit surface code patches, each encoding a logical qubit. We will use $\bar{X}_1$ and $\bar{X}_2$ to denote the Pauli X logical operators of these qubits, and $\bar{Z}_1$ and $\bar{Z}_2$ to denote their Pauli Z logical operators. Similarly, $\bar{X}_3$ and $\bar{Z}_3$ will denote the Pauli logical operators of the logical qubit resulting from the merge.

Rough merge. A rough merge is performed along the rough boundaries of the patches. It is realized by activating the surface code stabilizer measurements between the two rough boundaries of the patches. An illustration of a rough merge is provided in Figure 2.5. During this process, the weight-2 Z stabilizers along the boundaries are merged to form weight-4 Z stabilizers. Note that no new information is gained via these weight-4 Z stabilizers as they correspond to products of the initial boundaries' weight-2 stabilizers. The $\bar{Z}_3$ operator corresponds to any Z-chain connecting the rough boundaries and is equivalent to $\bar{Z}_1\bar{Z}_2$.

The X-type stabilizers in between the two patches are also activated. The product of these stabilizers includes all the qubits located at the rough boundaries on which the merge is being performed. Therefore, the product of these stabilizers is equal to $\bar{X}_1\bar{X}_2$. As such, the sum of the measurement outcomes of these X stabilizers corresponds to the outcome of a $\bar{X}_1\bar{X}_2$ measurement. The result of this measurement tells us whether or not $\bar{X}_1$ differs from $\bar{X}_2$. If the result of the joint measurement is $+1$, then $\bar{X}_1$ and $\bar{X}_2$ are equivalent. In this case, the Pauli X logical operator $\bar{X}_3$ of the logical qubit resulting from the merge is also equivalent to $\bar{X}_1$ and $\bar{X}_2$ and corresponds to any X-chain connecting the smooth boundaries of the newly formed patch. This rough merge operation with a $+1$ measurement outcome corresponds to the following map:

$$\mathcal{M}_+ = |+\rangle\langle++| + |-\rangle\langle--|\,. \tag{2.34}$$

This map can be interpreted as a CNOT operation followed by an X measurement on the controlled qubit which is post-selected to the $|+\rangle$ state.

If the result of the $\bar{X}_1\bar{X}_2$ measurement is -1, then $\bar{X}_1$ differs from $\bar{X}_2$. This implies that there is a Z-chain connecting the measured X-type stabilizers to one of the rough boundaries of the newly formed patch. To put the surface code back in a quiescent state, the endpoint of this Z-chain must be connected to one of the rough boundaries. The choice of the boundary implies whether $\bar{X}_1$ or $\bar{X}_2$ will be chosen to be equivalent to $\bar{X}_3$. This rough merge operation with a -1 measurement outcome corresponds to the following map:

$$\mathcal{M}_- = |+\rangle\langle+-| + |-\rangle\langle-+| \tag{2.35}$$

if $\bar{X}_3$ is chosen to be equivalent to $\bar{X}_1$, or to the following map:

$$\mathcal{M}_- = |+\rangle\langle-+| + |-\rangle\langle+-| \tag{2.36}$$

if $\bar{X}_3$ is chosen to be equivalent to $\bar{X}_2$.

Smooth merge. A smooth merge is performed along the smooth boundaries of the patches, analogous to the rough merge but with the roles of X and Z operators exchanged. The operation is realized by activating the surface code stabilizer measurements between the two smooth boundaries of the patches.

The $\bar{X}_3$ operator corresponds to any X-chain connecting the smooth boundaries and is equivalent to $\bar{X}_1\bar{X}_2$. The product of the new Z-type stabilizers is equal to $\bar{Z}_1\bar{Z}_2$. The result of this measurement tells us whether or not $\bar{Z}_1$ differs from $\bar{Z}_2$. If the result of the joint measurement is $+1$, then $\bar{Z}_1$ and $\bar{Z}_2$ are equivalent. In this case, the Pauli Z logical operator $\bar{Z}_3$ of the logical qubit resulting from the merge is also equivalent to $\bar{Z}_1$ and $\bar{Z}_2$ and corresponds to any Z-chain connecting the rough boundaries of the newly formed patch. This smooth merge operation with a $+1$ measurement outcome corresponds to the following map:

$$\mathcal{M}_+ = |0\rangle\langle00| + |1\rangle\langle11| \tag{2.37}$$

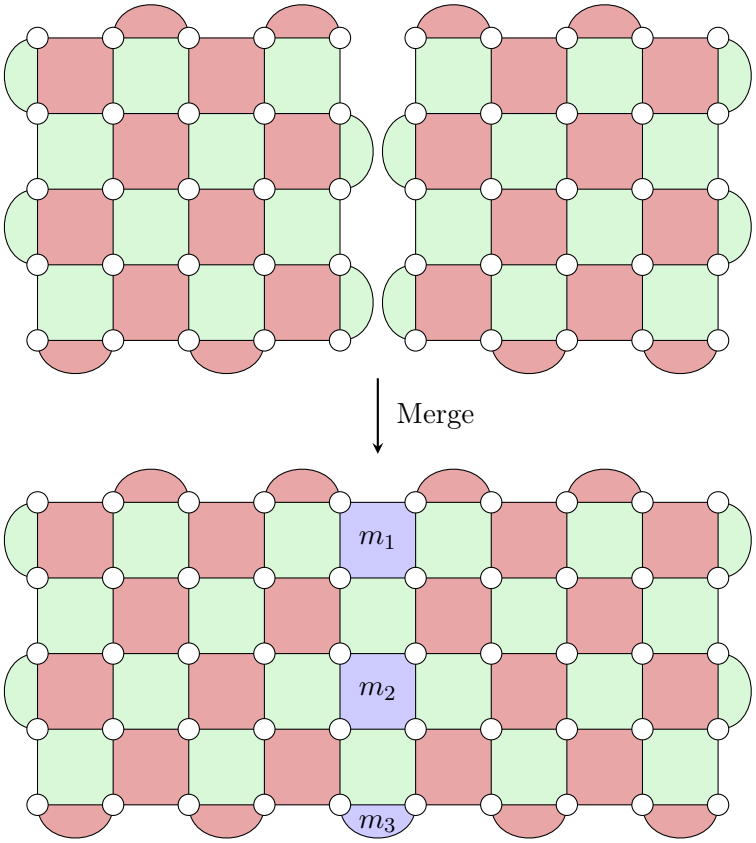


Figure 2.5: Example of a lattice surgery rough merge operation between two surface code logical qubits. The initial layout at the top shows two 5×5 surface code patches with adjacent rough boundaries. The bottom illustration represent the patch resulting from the merge operation. The blue regions (m_1, m_2, m_3) are indicating X stabilizer measurements with random outcomes.

This map can be interpreted as a CNOT operation followed by a Z measurement on the target qubit which is post-selected to the $|0\rangle$ state.

The case where the result of the $\bar{Z}_1\bar{Z}_2$ measurement is -1 is also analogous to the rough merge. The smooth merge operation with a -1 measurement outcome corresponds to the following map:

$$\mathcal{M}_- = |0\rangle\langle 01| + |1\rangle\langle 10| \tag{2.38}$$

if $\bar{Z}_3$ is chosen to be equivalent to $\bar{Z}_1$, or to the following map:

$$\mathcal{M}_- = |0\rangle\langle 10| + |1\rangle\langle 01| \tag{2.39}$$

if $\bar{Z}_3$ is chosen to be equivalent to $\bar{Z}_2$.

2.2.1.2 Split operation

The split operation is the converse of the merge operation, where a single logical qubit is divided into two separate logical qubits. This operation is also performed along either rough or smooth boundaries. We will use $\bar{X}_1$ and $\bar{Z}_1$ to denote the Pauli X and Pauli Z logical operators of the logical qubit on which the split operation is performed. Similarly, we will use $\bar{X}_2$, $\bar{X}_3$ and $\bar{Z}_2$, $\bar{Z}_3$ to denote the Pauli logical operators of the logical qubits resulting from the split.

Rough split. A rough split of a single logical qubit is performed parallel to the rough boundaries of the surface code patch. It consists in deactivating some stabilizer measurements to separate the patch into two distinct patches. An illustration of a rough split is provided in Figure 2.6.

The changes in the stabilizer measurements are the reverse of the changes done for the rough merge operation. The weight-4 Z stabilizers are split back into weight-2 Z stabilizers. The X-type stabilizers in between the two patches anticommute with the newly introduced weight-2 Z stabilizer and are removed from the stabilizer group. The $\bar{X}_2$ and $\bar{X}_3$ logical operators are both commuting with the weight-2 Z-type stabilizers measurement and are equivalent to $\bar{X}_1$. The $\bar{Z}_1$ logical operator is split in two and is equivalent to $\bar{Z}_2\bar{Z}_3$.

In the case where some of the newly introduced weight-2 Z stabilizers measurements have an outcome of -1, corrections are needed to put the code back into a quiescent state. We can notice that each pair of facing weight-2 Z stabilizers have a correlated measurement outcome as the qubits involved in these stabilizer measurements were part of a weight-4 Z stabilizer before the split. For each pair with a measurement outcome of -1, a correction can then be realized by using two X-chains connecting these weight-2 stabilizers to the same smooth boundary. The resulting operation is either a trivial logical operation or is equal to $\bar{X}_2\bar{X}_3$, which is equivalent to $\bar{X}_1\bar{X}_1$ and is therefore equal to the identity operation.

This rough split operation corresponds to the following map:

$$\mathcal{S} = |{+}{+}\rangle\langle{+}| + |{-}{-}\rangle\langle{-}| \tag{2.40}$$

This map can be interpreted as a CNOT operation where the initial qubit is the target qubit and the control qubit is initialized in the $|+\rangle$ state.

Smooth split. A smooth split is analogous to the rough split but with the roles of X and Z operators and stabilizers exchanged. It is performed parallel to the smooth boundaries of the surface code patch. The changes in the stabilizer measurements are the reverse of the changes done for the smooth merge operation. The weight-4 X stabilizers are split back into weight-2 X stabilizers. The $\bar{Z}_2$ and $\bar{Z}_3$ logical operators are both commuting with the weight-2 X-type stabilizers measurement and are equivalent to $\bar{Z}_1$. The $\bar{X}_1$ operator is split in two and is equivalent to $\bar{X}_2\bar{X}_3$.

Analogously to the rough merge, for each pair of weight-2 X stabilizers with measurement outcome -1, a correction can be realized by using two Z-chains connecting these weight-2 stabilizers to the same rough boundary. The resulting operation is either a trivial logical operation or is equal to $\bar{Z}_2\bar{Z}_3$, which is equivalent to $\bar{Z}_1\bar{Z}_1$ and is therefore equal to the identity operation.

This smooth split operation corresponds to the following map:

$$\mathcal{S} = |00\rangle\langle 0| + |11\rangle\langle 1| \tag{2.41}$$

This map can be interpreted as a CNOT operation where the initial qubit is the control qubit and the target qubit is initialized in the $|0\rangle$ state.

2.2.2 Magic state distillation

While Clifford gates can be implemented fault-tolerantly in the surface code through efficient techniques such as lattice surgery, the fault-tolerant implementation of the T gate requires fundamentally different approaches such as magic state distillation [38]. A magic state is a quantum state that, when combined with Clifford operations, enables the implementation of non-Clifford

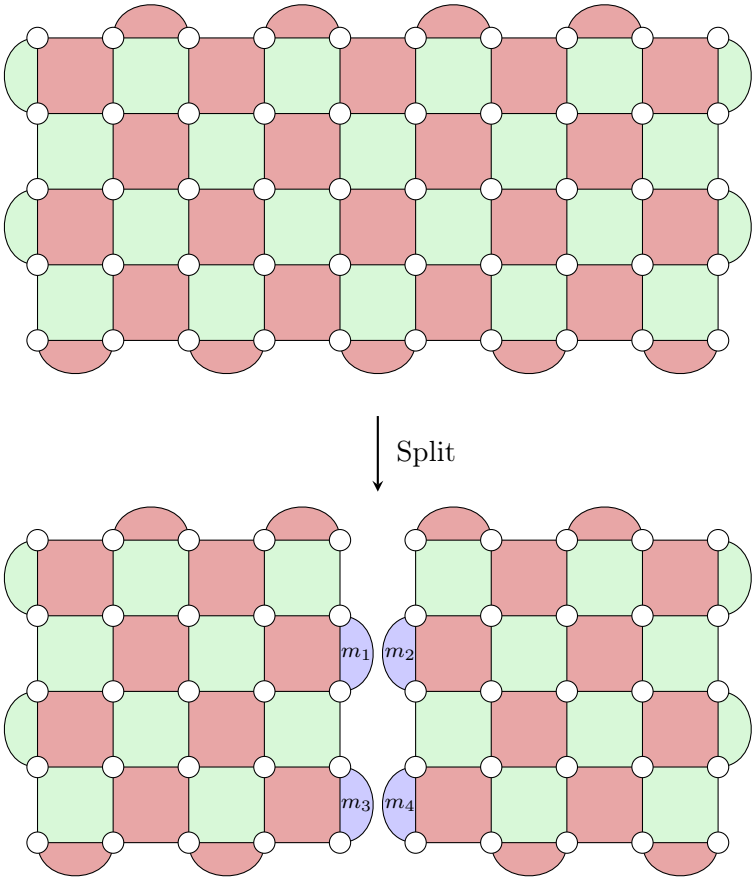


Figure 2.6: Example of a lattice surgery rough split operation between two surface code logical qubits. The initial layout at the top shows a 5×10 surface code patch. The bottom illustration represent the patches resulting from the split operation. The blue regions (m_1, m_2, m_3, m_4) are indicating Z stabilizer measurements with random outcomes.

gates such as the T gate. Magic state distillation consists in purifying noisy magic states to produce high-fidelity magic states, which can then be used to implement the T gate fault-tolerantly.

Magic state distillation protocols are more costly to implement than Clifford gates, both in terms of the number of physical qubits required and the time needed for implementation. This overhead motivates the optimization of T gates in quantum circuits, as each T gate will require the consumption of a magic state.

2.2.2.1 Magic states

A magic state for the T gate, denoted as $|m\rangle$, is defined as:

$$|m\rangle = T\,|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{i\pi/4}\,|1\rangle) \tag{2.42}$$

Such a magic state can be used to implement a T gate via a gate teleportation circuit, as depicted in Figure 2.7.

Magic states can be created in the surface code by a process called state injection. However, this process is not fault-tolerant as it can lead to faulty magic states. A faulty magic state

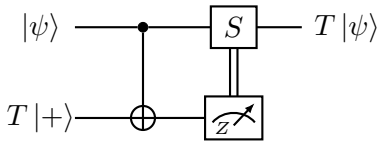


Figure 2.7: Teleportation circuit to implement a T gate using a magic state.

corresponds to a magic state that has experienced an undesired Pauli Z rotation, resulting in the state $Z|m\rangle$. During magic state injection, such error can occur with some probability p. The goal of magic state distillation is to purify the noisy magic states resulting from state injection to produce high-fidelity magic states that can be reliably used for quantum computation.

2.2.2.2 Triorthogonal codes

A family of magic state distillation protocols can be defined by triorthogonal matrices [60]. A binary matrix P of size $n \times m$ is called triorthogonal if it satisfies the following equations for all indices i, j, k such that $0 < i < j < k < n$:

$$|P_i \wedge P_j| \equiv 0 \pmod 2 \tag{2.43}$$

$$|P_i \wedge P_j \wedge P_k| \equiv 0 \pmod 2 \tag{2.44}$$

where P_i is the ith row of P, $|P_i|$ denotes its Hamming weight, and the $\wedge$ operator represents the bitwise AND operation.

Let P be a triorthogonal matrix. Each column of P represents a Pauli Z rotation of angle $\pi/4$. As such, the number m of noisy magic states consumed by the magic state distillation protocol represented by P is equal to the number of columns of the triorthogonal matrix. And the number k of distilled magic states produced by P is equal to the number of rows of P satisfying

$$|P_i| \equiv 1 \pmod 2. \tag{2.45}$$

The other rows of P satisfying

$$|P_i| \equiv 0 \pmod 2 \tag{2.46}$$

are associated with the qubits of the code which are measured to detect faulty magic states. Without any error, measuring these qubits at the end of the circuit must result in a $+1$ measurement outcome. However, if a faulty magic state is used instead of a perfect one, then it propagates the error on all the qubits on which the Pauli rotation is acting non-trivially, which would lead to a measurement outcome of -1 for the qubits being measured, effectively detecting the presence of an error.

The code distance d of a triorthogonal matrix P is the minimal number of errors that the code fails to detect. When an error is detected, two strategies are possible. The first approach is to attempt error correction, which can fail whenever the number of errors is greater than $\lfloor d/2 \rfloor$. The error probability of the magic state produced by this method has a probability of being faulty equal to $\mathcal{O}(p^{d/2})$ where p is the probability for each input magic state to be faulty. Another approach is to discard the produced magic state and restart the distillation process. When this method succeeds, it leads to a magic state with a probability of being faulty equal to $\mathcal{O}(p^d)$.

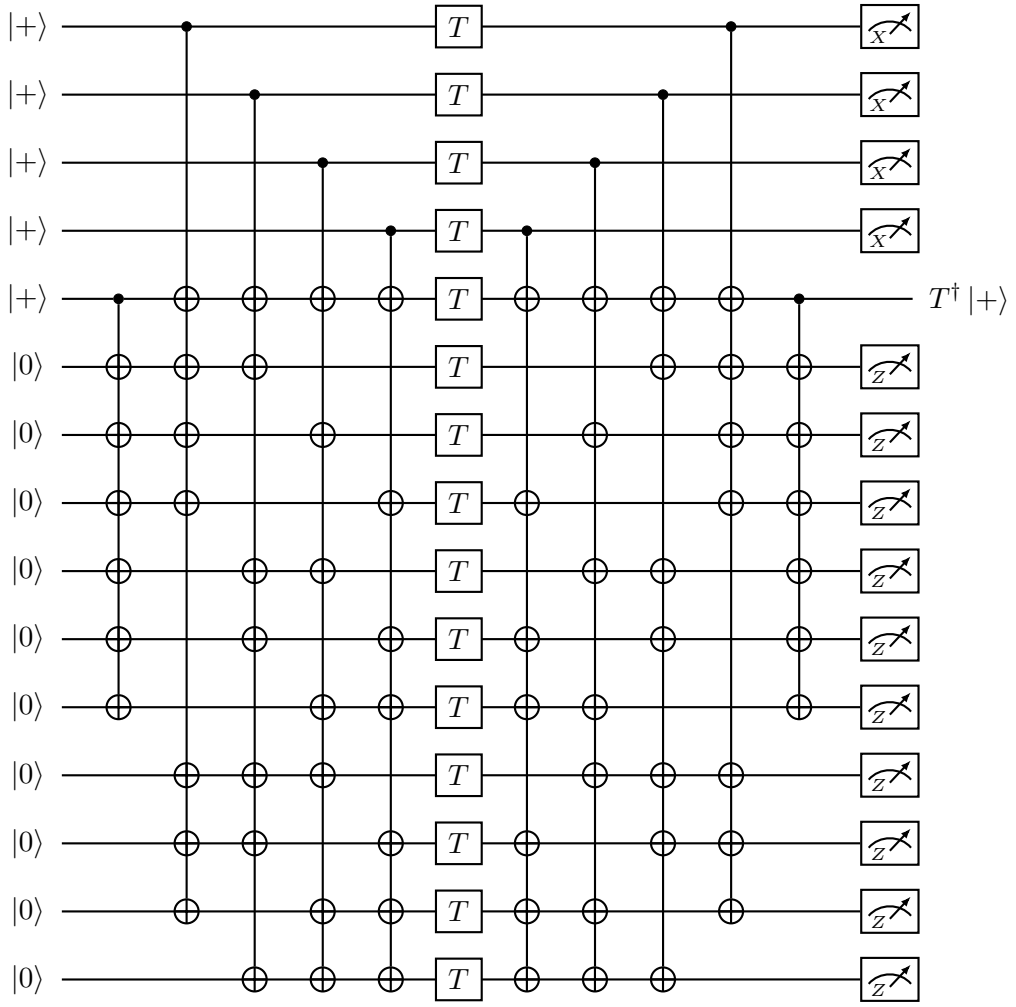


Figure 2.8: Quantum circuit for the 15-to-1 magic state distillation protocol.

For example, the following triorthogonal matrix describes the 15-to-1 distillation protocol:

$$P = \left(\begin{array}{cccccccccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right). \tag{2.47}$$

This protocol consumes 15 noisy magic states to produce one magic state of higher fidelity. The Hamming weight of the first row of P is odd and is associated with the qubit storing the resulting magic state. The four last rows of P have an even Hamming weight, corresponding to the qubits of the code being measured to detect errors. This code always detects an error whenever fewer than three input magic states are faulty. However, it fails to detect all combinations of three faulty input magic states. For instance, the first, second, and seventh columns form such a combination, which means that faulty magic states used for these three Pauli rotations will not lead to an error detection. There are 35 such combinations of undetected three faulty input magic states, which is why the probability of the output magic state being faulty is about $35p^3$

when the magic state is discarded whenever an error is detected. A circuit implementing the 15-to-1 magic state distillation protocol is presented in Figure 2.8.

Chapter 3

The ZX-Calculus

The ZX-calculus is a graphical language designed for reasoning about quantum computation. Introduced by Bob Coecke and Ross Duncan in 2008 [61], it emerged from the broader framework of categorical quantum mechanics [62].

The ZX-calculus represents quantum computation over qubits through a simple yet expressive diagrammatic notation. Analogously to quantum circuits, it uses wires to represent quantum information flow. However, while circuit gates must adhere to unitarity, the operational components of the ZX-calculus are free from this constraint. In this sense, the ZX-calculus can be regarded as a more expressive generalization of quantum circuits.

The ZX-calculus is equipped with a set of graphical rewriting rules that enable rigorous manipulation of ZX-diagrams, the graphical elements of the language. A valuable property of the ZX-calculus is its completeness for quantum mechanics. Completeness implies that any two ZX-diagrams representing the same quantum evolution can be transformed into one another using a given set of rewriting rules. The completeness of the ZX-calculus was initially proven for restricted subsets of quantum mechanics, such as stabilizer quantum mechanics [63]. It has then been proven for the widely used approximately universal Clifford+T fragment of quantum mechanics [64], and for universal quantum mechanics [65]. In principle, this allows all reasoning about quantum computation to be performed within the calculus, making it a self-contained language.

Several quantum computing applications of the ZX-calculus have been developed. One of its first notable applications was in measurement-based quantum computation (MBQC), where it provides an intuitive framework for representing graph states and deriving measurement patterns [66]. Moreover, the simple and intuitive rewriting rules of the ZX-calculus have been used to design automated optimization procedures for quantum circuits [67, 68], and for quantum circuit simulation [69, 70].

Additionally, the ZX-calculus has been used to help with the visualization and manual optimization of some fault-tolerant quantum computing protocols [71–73]. Notably, it has been proven that ZX-calculus is a language for surface code lattice surgery [74]. While quantum circuits are not well-suited for representing lattice surgery operations directly, the ZX-calculus provides a natural language for this purpose. This enables the use of ZX-calculus to characterize determinism in surface code lattice surgery [75], and to develop optimization methods for lattice surgery such as done in this thesis.

In this chapter, we provide an introduction to the ZX-calculus by presenting ZX-diagrams and the common rules of the calculus. Then, we show how the surface code lattice surgery operations can be naturally described by the ZX-calculus.

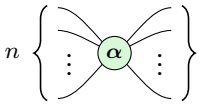
3.1 ZX-diagrams

3.1.1 Nodes and wires

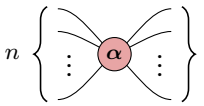
The graphical components of ZX-diagrams are nodes and wires. The nodes are either green () or red (), and they have an associated angle, also referred to as the phase and here denoted by α, which can be omitted when it is equal to 0. The wires serve a similar role to the horizontal lines in quantum circuits, representing the flow of quantum information between nodes of the diagram.

3.1.2 Spiders

A spider is the fundamental operational element in ZX-calculus. It is a green or red node that can have incident wires, referred to as input wires of the spider if they are attached to the left side of the node or output wires of the spider if they are attached to the right side of the node. It represents the following linear maps:



$$:= |0\rangle^{\otimes m}\langle 0|^{\otimes n} + e^{i\alpha}|1\rangle^{\otimes m}\langle 1|^{\otimes n} \tag{3.1}$$



$$:= |+\rangle^{\otimes m}\langle +|^{\otimes n} + e^{i\alpha}|-\rangle^{\otimes m}\langle -|^{\otimes n} \tag{3.2}$$

Green node spiders are called Z spiders and red node spiders are called X spiders.

3.1.3 Structure of ZX-diagrams

A ZX-diagram is composed of wires and spiders. Vertical wires can be interpreted as either bent to the right or to the left; it doesn't alter the overall interpretation of the diagram. A wire is said to be an open wire if at least one of its endpoints is not connected to a node. For a given ZX-diagram $\mathcal{D}$, an open wire is an input wire of $\mathcal{D}$ if it has at least one open endpoint pointing towards the left, and it is an output wire of $\mathcal{D}$ if it has at least one open endpoint pointing towards the right. For clarity, we will always put the open endpoint of input wires at the far left of the diagram (before all nodes), and the open endpoint of output wires at the far right of the diagram (after all nodes), as commonly done for quantum circuits.

ZX-diagrams that do not have any input or output wires are scalars, representing renormalization of quantum states. For practical purposes, we will ignore all scalar factors and write all ZX-diagram equalities up to a non-zero scalar.

Given two diagrams $\mathcal{D}_1$ and $\mathcal{D}_2$, we can compose them sequentially by connecting the output wires of $\mathcal{D}_1$ to the input wires of $\mathcal{D}_2$, corresponding to the sequential application of operations. Diagrams can also be composed vertically using the tensor product operation, written as $\mathcal{D}_1 \otimes \mathcal{D}_2$, representing parallel operations. For example, the following three elementary ZX-diagrams:

$$\mathcal{D}_1 = \quad = \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}, \tag{3.3}$$

$$\mathcal{D}_2 = \quad = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \tag{3.4}$$

$$\mathcal{D}_3 = \quad = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}. \tag{3.5}$$

can be composed as $(\mathcal{D}_1 \otimes \mathcal{D}_2)(\mathcal{D}_2 \otimes \mathcal{D}_3)$ to build a ZX-diagram equivalent to the CNOT gate up to a scalar factor:

$$\begin{aligned} &= \left(\begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right) \left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \right) \\ &= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \text{CNOT}. \end{aligned} \tag{3.6}$$

3.2 Rules of the ZX-calculus

The ZX-calculus has many different diagram rewriting rules for transforming a ZX-diagram into an equivalent one. In this section, we introduce some of the most fundamental and common rules of the ZX-calculus.

3.2.1 Only connectivity matters

The first one is a meta-rule: if two ZX-diagrams have the same open wires and the same set of spiders connected in the same manner, then these two diagrams are equivalent. For example, these two ZX-diagrams are equivalent:

$$\quad = \tag{3.7}$$

This rule is particularly useful as it allows us to treat a family of equivalent ZX-diagrams in a more abstract way by relying on undirected graphs (with additional parameters associated with the vertices).

3.2.2 Identity rule

The identity rule states that a wire is equivalent to a spider with no phase and two incident wires:

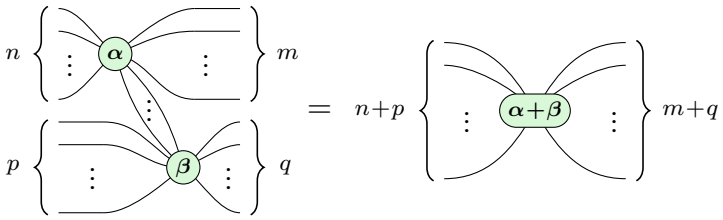
$$\quad = \quad = \tag{3.8}$$

This rule provides a simple way of representing the Bell state $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle+|11\rangle) = \frac{1}{\sqrt{2}}(|++\rangle+|--\rangle)$ in ZX-calculus by a single wire bent such that both endpoints are outputs:

$$\begin{pmatrix}1\\0\\0\\1\end{pmatrix} = \quad = \quad = \tag{3.9}$$

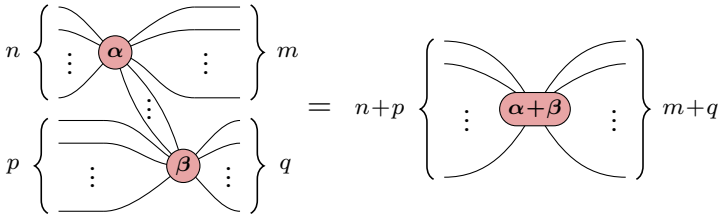
3.2.3 Spider fusion

If two Z spiders are connected by at least one wire, then they can be fused together into a single spider, and the resulting phase is the sum of their individual phases:



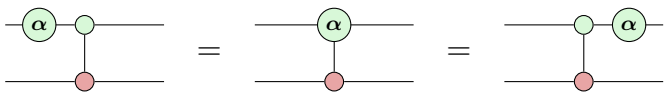
(3.10)

This spider fusion rule also holds for X spiders, as every rule of the ZX-calculus remains true when swapping Z and X spiders:



(3.11)

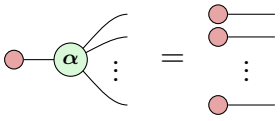
The spider fusion rule facilitates several intuitive transformations, such as the commutation of some quantum gates. For instance, an R_Z rotation commutes with the control of a CNOT gate:



(3.12)

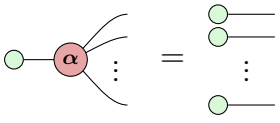
3.2.4 Copy and bialgebra rules

The ZX-calculus also incorporates rules centered around the interactions between green and red spiders. The copy rule states that a Z spider copies X spiders of arity one:



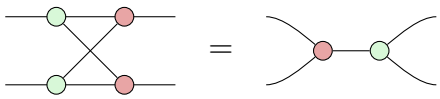
(3.13)

Similarly, an X spider copies Z spiders of arity one:



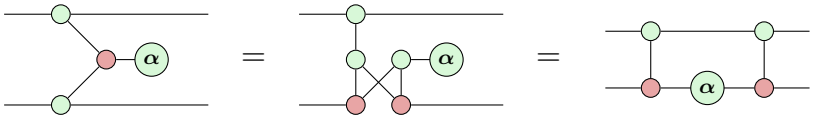
(3.14)

Another crucial interaction between green and red spiders is given by the bialgebra rule:



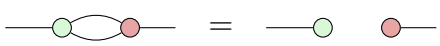
(3.15)

For instance, the following phase gadget equality can be proven using the bialgebra and fusion rules:



(3.16)

Moreover, the copy and bialgebra rules, together with the identity and spider fusion rules, can be used to derive the following equality, known as the Hopf rule:



(3.17)

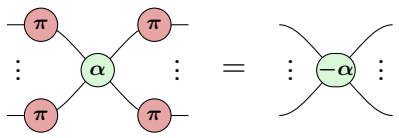
3.2.5 Node transformation rules

Applying a Clifford gate to a stabilizer state always results in another stabilizer state. Given that a spider is either defined over the computational basis stabilizer states (for green spiders) or the diagonal basis stabilizer states (for red spiders), conjugating a spider by a Clifford gate that performs a mapping between the stabilizer states $|0\rangle^{\otimes n}$, $|1\rangle^{\otimes n}$, $|+\rangle^{\otimes n}$, and $|-\rangle^{\otimes n}$ will result in another spider with only its node potentially changed.

For example, the Pauli X gate maps the stabilizer state $|0\rangle$ to the stabilizer state $|1\rangle$, and vice versa. Therefore, we have:

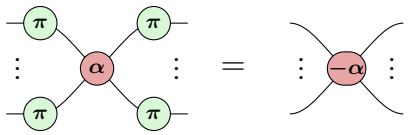
$$\begin{aligned} & X^{\otimes m}\left(|0\rangle^{\otimes m}\langle 0|^{\otimes n}+e^{i\alpha}|1\rangle^{\otimes m}\langle 1|^{\otimes n}\right)X^{\otimes n} \\ &= X^{\otimes m}|0\rangle^{\otimes m}\langle 0|^{\otimes n}X^{\otimes n}+e^{i\alpha}X^{\otimes m}|1\rangle^{\otimes m}\langle 1|^{\otimes n}X^{\otimes n} \\ &= |1\rangle^{\otimes m}\langle 1|^{\otimes n}+e^{i\alpha}|0\rangle^{\otimes m}\langle 0|^{\otimes n} \\ &= e^{-i\alpha}|1\rangle^{\otimes m}\langle 1|^{\otimes n}+|0\rangle^{\otimes m}\langle 0|^{\otimes n} \end{aligned} \tag{3.18}$$

which implies the following equality:



(3.19)

An analogous rule can be defined for the red spider:



(3.20)

Similarly, conjugating a green spider with the Hadamard gate H (which maps the stabilizer states $|0\rangle$ to $|+\rangle$ and $|1\rangle$ to $|-\rangle$, and vice versa) yields:

$$\begin{aligned} & H^{\otimes m}\left(|0\rangle^{\otimes m}\langle 0|^{\otimes n}+e^{i\alpha}|1\rangle^{\otimes m}\langle 1|^{\otimes n}\right)H^{\otimes n} \\ &= H^{\otimes m}|0\rangle^{\otimes m}\langle 0|^{\otimes n}H^{\otimes n}+e^{i\alpha}H^{\otimes m}|1\rangle^{\otimes m}\langle 1|^{\otimes n}H^{\otimes n} \\ &= |+\rangle^{\otimes m}\langle +|^{\otimes n}+e^{i\alpha}|-\rangle^{\otimes m}\langle -|^{\otimes n} \end{aligned} \tag{3.21}$$

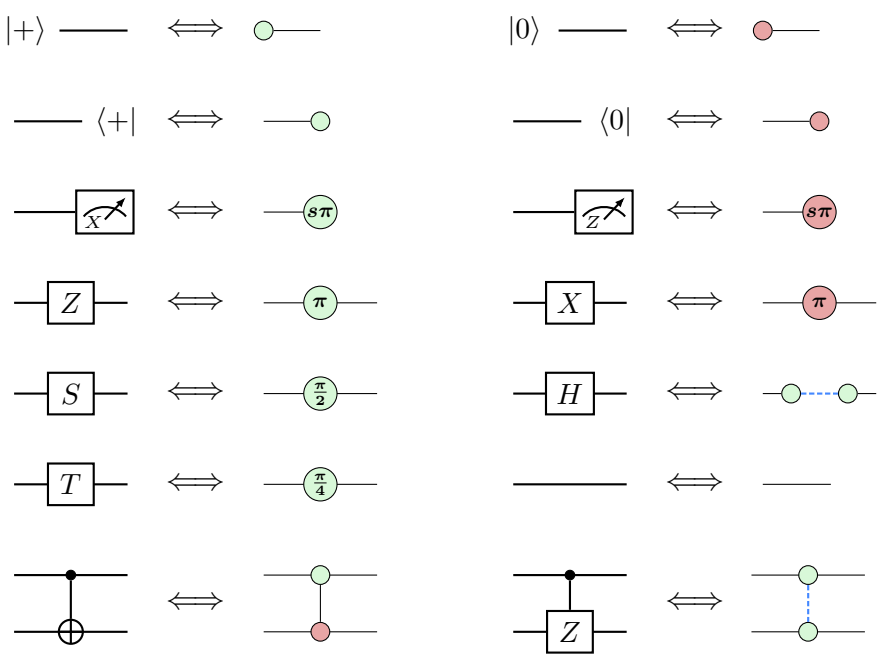
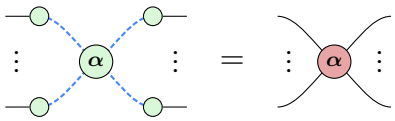


Figure 3.1: Translation table for quantum circuits and ZX-diagrams. Measurements in quantum circuits are represented as projections in ZX-diagrams, with a conditional π-rotation based on the measurement outcome represented by the variable $s \in \{0, 1\}$.

which corresponds to the definition of a red spider, and therefore proves the following color change rule:



(3.22)

This rule, combined with the fact that two adjacent Hadamard edges are equal to the identity:

(3.23)

can be used to prove that every rule of the ZX-calculus remains true when the colors of the spiders are swapped.

3.3 Quantum gates in the ZX-calculus

3.3.1 From quantum circuits to ZX-diagrams

We can easily translate a quantum circuit composed of CNOT gates and R_Z (or R_X) gates into a ZX-diagram by using the following procedure:

- Substitute all R_Z rotations (respectively, R_X rotations) of angle α with green nodes (respectively, red nodes) having the same angle α.
- Substitute all qubits initialization in the $|+\rangle$ state (respectively, $|0\rangle$ state) by green nodes (respectively, red nodes).

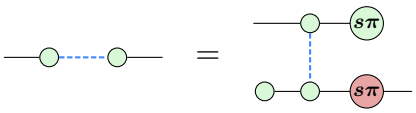
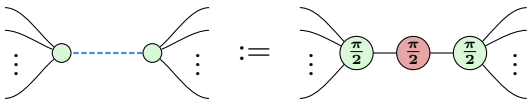


Figure 3.2: Hadamard gate gadgetization in ZX-calculus, where the variable $s \in \{0, 1\}$ depends on the measurement outcome.

- For each CNOT gate, replace the control node with a green node and the target node with a red node.
- Keep the same connectivity.

For a Clifford+T circuit, the associated ZX-diagram can be constructed by using the translation table represented in Figure 3.1. For example, the Hadamard gate gadgetization circuit presented in Figure 1.2 corresponds to the ZX-diagram depicted in Figure 3.2.

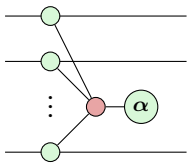
As a shorthand notation, a Hadamard gate is often represented by a dashed blue edge. But it can also be expressed using green and red spiders through the following equality:



(3.24)

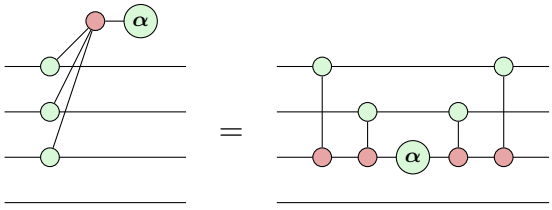
3.3.2 Phase and Pauli gadgets

Diagonal Pauli rotations ($R_P(\alpha)$ gates where $P_i \in \{I, Z\}$ for all i) can be represented in the ZX-calculus using phase gadgets:



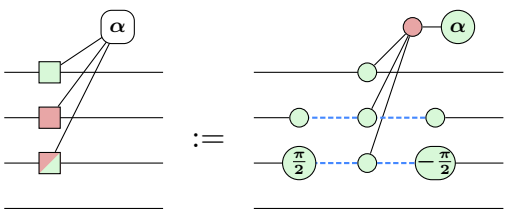
(3.25)

For instance, the Pauli rotation $R_{ZZZI}(\alpha)$ corresponds to the following phase gadget, which can be transformed into a circuit-like form using CNOT gates and an $R_Z(\alpha)$ gate as follows:



(3.26)

Phase gadgets can be generalized to Pauli gadgets, which can represent any Pauli rotation. To facilitate this, we introduce a shorthand notation: green, red, and bicolor boxes which are associated with Pauli Z, X, and Y, respectively. For example, the Pauli rotation $R_{ZXYI}(\alpha)$ can be represented in the ZX-calculus using the following Pauli gadget:



(3.27)

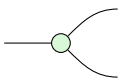
Pauli gadgets are notably useful for representing Pauli rotation sequences, defined in Section 1.2.3.4.

3.4 The ZX-calculus as a language for surface code lattice surgery

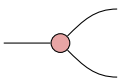
The surface code lattice surgery is a measurement-based method for performing fault-tolerant logical operations, as described in Section 2.2.1. However, these logical operations are not easily captured by quantum circuit model. Also, some of these operations are non-deterministic, meaning they may require additional steps to achieve the desired logical transformation. In this section, we will show how these problems can be addressed by using the ZX-calculus as a language for surface code lattice surgery.

3.4.1 Lattice surgery operations in ZX-calculus

The fundamental lattice surgery operations are the split and merge operations. The split operation consists in splitting a logical qubit into several logical qubits, whereas the merge operation consists in merging several logical qubits into a single logical qubit. For the surface code lattice surgery, the split and merge operations can be divided into two types: smooth or rough. This distinction is based on the type of logical operator on which the operation acts non-trivially. It has been shown that the ZX-calculus is a language for the surface code lattice surgery [74], where the wires of a ZX-diagram are associated with logical qubits and the spiders are representing the lattice surgery operations acting on these logical qubits. The red spiders are representing rough operations, while the green spiders are representing smooth operations. The split operation is represented by a spider with one input wire and multiple output wires. For example, the smooth and rough splits of a single logical qubit into two logical qubits are represented by a green and red spider respectively, with one input wire and two output wires:

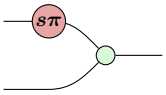


Smooth split

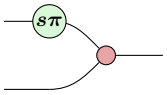


Rough split

The merge operation can be seen as the converse of the split operation. It is represented by a spider with multiple input wires and one output wire. However, unlike the split operation, the merge operation is non-deterministic. Pauli errors, which are correlated with the outcome of the measurements involved in realizing this operation, may be introduced on all but one of the input wires. For example, the smooth and rough merges of two logical qubits are represented by a green and red spider, respectively, with two input wires and one output wire, and where a Pauli error may occur on one of the input wires:



Smooth merge



Rough merge

where $s \in \{0, 1\}$. The value of s is known only once the merge has been realized, which makes the merge operation non-deterministic by nature.

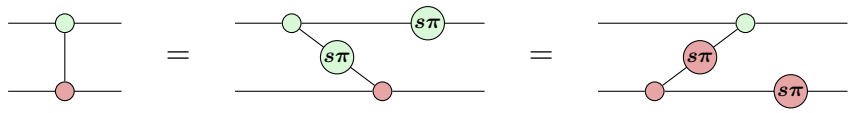
Projections in surface code lattice surgery are also non-deterministic, as they are realized via measurements. These operations can be naturally represented in ZX-calculus as follows:

X-projection

Z-projection

where $s \in \{0, 1\}$.

Some sequences of lattice surgery operations can be realized deterministically even if they include some non-deterministic merge operations. To achieve this, a correction must be found for each of the potential Pauli errors. A correction involves modifying the subsequent operations to cancel out the effect of the Pauli error. For instance, the CNOT gate can be implemented using lattice surgery operations as follows:



where $s \in \{0, 1\}$.

3.4.2 Pauli Fusion flow

An algorithm for finding a correction strategy, called Pauli Fusion flow (PF-flow), for a given ZX-diagram representing lattice surgery operations was proposed in Reference [75]. A PF-flow preserves the connectivity of the diagram and may change only the phases of the spiders by changing the sign of the angle or adding π. To define the PF-flow, we first need to introduce several concepts: graph-like ZX-diagrams, the signature of a ZX-diagram, and the notion of a corrector.

Graph-like ZX-diagram. A ZX-diagram is in graph-like form if it satisfies the following conditions:

1. It has no connections (including self-loops) between spiders of the same color.
2. It has no more than one wire connecting any pair of spiders.

Any ZX-diagram can be transformed into a graph-like ZX-diagram by using the spider fusion rule to satisfy the first condition, and the Hopf rule to satisfy the second condition. The PF-flow is designed for graph-like ZX-diagrams. This is without loss of generality, as a PF-flow for a graph-like ZX-diagram can be used to derive an equivalent correction strategy for the original ZX-diagram.

ZX-diagram signature. For a graph-like ZX-diagram $\mathcal{D}$, the signature graph $\mathcal{G}_\mathcal{D}$ is an undirected graph obtained through the following construction:

1. Convert each spider into a graph vertex while maintaining the same connectivity.
2. For each input wire of $\mathcal{D}$, add a vertex to the open end.
3. For each output wire of $\mathcal{D}$, add a vertex to the open end.
4. Add a self-loop to each vertex associated with a spider in $\mathcal{D}$ whose phase is an odd multiple of $\pi/2$.

An example of a signature graph is provided in Figure 3.3. The signature of $\mathcal{D}$ is then defined as a tuple $(\mathcal{G}_\mathcal{D}, \mathcal{I}, \mathcal{O}, \mathcal{P})$ where:

- $\mathcal{I}$ are the vertices in $\mathcal{G}_\mathcal{D}$ added to the open ends of input wires.
- $\mathcal{O}$ are the vertices in $\mathcal{G}_\mathcal{D}$ added to the open ends of output wires.

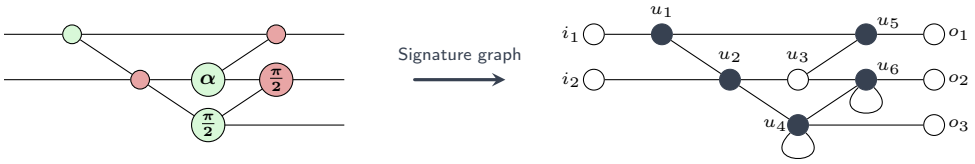


Figure 3.3: A graph-like ZX-diagram $\mathcal{D}$ and its associated signature graph $\mathcal{G}_\mathcal{D}$. The input vertices are $\mathcal{I} = \{i_1, i_2\}$ and the output vertices are $\mathcal{O} = \{o_1, o_2, o_3\}$. The set of vertices $\mathcal{P}$ associated with spiders whose phases are multiples of $\pi/2$ are shown in black.

- $\mathcal{P}$ are the vertices in $\mathcal{G}_\mathcal{D}$ associated with spiders whose phases are multiples of $\pi/2$.

The signature of a graph-like ZX-diagram captures the essential structural properties needed for constructing a PF-flow.

Corrector. Let $\mathcal{D}$ be a graph-like ZX-diagram, $\mathcal{G}_\mathcal{D}$ be its signature graph, $V(\mathcal{G}_\mathcal{D})$ be the vertex set of $\mathcal{G}_\mathcal{D}$, and let $\preccurlyeq$ be a partial order on $V(\mathcal{G}_\mathcal{D})$ (representing an ordering of the operations of $\mathcal{D}$). For vertices $u, v \in V(\mathcal{G}_\mathcal{D})$ and a subset $C \subseteq V(\mathcal{G}_\mathcal{D})$, we say that C is a v-corrector of u if:

1. $u \in \mathrm{Odd}(C)$ (where $\mathrm{Odd}(C)$ is the set of vertices adjacent to an odd number of elements in C).
2. $v \prec w$ for all $w \in (C \setminus \mathcal{P}) \cup (\mathrm{Odd}(C) \setminus \{u\})$.

where $v \prec w$ indicates that v strictly precedes w in the partial order $\preccurlyeq$ defined on $V(\mathcal{G}_D)$.

For example, a u_6-corrector of u_3 in the signature graph of Figure 3.3 is $\{u_6, o_2, o_3\}$, where vertices to the left precede those to the right. A v-corrector of u describes a mechanism for propagating a π-phase from a node u, which satisfies $u \prec v$, into operations that come after v. The π-phase propagation is realized by applying the rule of Equation 3.19 or Equation 3.20 to all the spiders associated with the vertices in the corrector C. The π-phase surrounding a spider w then cancel each other if and only if $w \notin \mathrm{Odd}(C)$. The $\pi/2$ cases are handled by the self-loop in the signature graph, which relates to the fact that the sign change in Equation 3.19 and Equation 3.20 introduces another π-phase because $-\pi/2 \equiv \pi/2 + \pi \pmod{2\pi}$.

Pauli Fusion flow. Let $\mathcal{D}$ be a graph-like ZX-diagram, $(\mathcal{G}_\mathcal{D}, \mathcal{I}, \mathcal{O}, \mathcal{P})$ be its signature, $V(\mathcal{G}_\mathcal{D})$ be the vertex set of $\mathcal{G}_\mathcal{D}$, $V(\mathcal{D}) = V(\mathcal{G}_\mathcal{D}) \setminus (\mathcal{I} \cup \mathcal{O})$, and let $\preccurlyeq$ be a partial order on $V(\mathcal{G}_\mathcal{D})$ (representing an ordering of the operations of $\mathcal{D}$). A PF-flow is a triple $(\preccurlyeq, f, \mathcal{C})$ consisting of a partial order $\preccurlyeq$ on $V(\mathcal{G}_\mathcal{D})$, a function $f : V(\mathcal{D}) \to V(\mathcal{G}_\mathcal{D})$, and a set $\mathcal{C}$ of corrector sets over $V(\mathcal{G}_\mathcal{D})$, which satisfies the following conditions for all $v \in V(\mathcal{D})$:

1. If vertices u, v are adjacent in $\mathcal{G}_\mathcal{D}$, then $u \prec v$ (and $u \notin \mathcal{O}$), or $v \prec u$ (and $u \notin \mathcal{I}$).
2. If there does not exist u such that $v \prec u$, then there exists a v-corrector of v in $\mathcal{C}$.
3. If vertices u, v are adjacent in $\mathcal{G}_\mathcal{D}$ such that $u \prec v$ and $f(v) \neq u$, there exists a v-corrector of u in $\mathcal{C}$.

where $u \prec v$ indicates that u strictly precedes v in the partial order $\preccurlyeq$, and the function f ensures that at most a single past neighbor of each vertex does not require a corrector. The first condition ensures that neighboring vertices have a defined ordering, and that all input vertices precede the vertices in $V(\mathcal{D})$ and all output vertices succeed the vertices in $V(\mathcal{D})$. The second

condition states that vertices that do not have any future neighbors require the existence of a v-corrector of v. This ensures that projection operations can be done deterministically. The third condition states that all past neighbors of v (except one, indicated by $f(v)$) must have a v-corrector. This ensures that merge operations can be done deterministically.

A polynomial-time algorithm for finding a PF-flow when one exists is given in Reference [75].

Part II

Quantum Circuit Synthesis of Arithmetic Operators

Chapter 4

Ancilla-Free Quantum Adder with Polylogarithmic Depth

Addition is a fundamental operation that serves as a building block for various quantum algorithms. For instance, modular exponentiation, which is a critical component of Shor's factoring algorithm [15], can be computed using modular multiplier circuits, which can be constructed using addition operations [76]. As such, the efficiency and scalability of quantum adders directly impact the performance of these higher-level algorithms.

There exists various approaches to design quantum circuits that perform the addition of two n-bit integers. Among these, ripple-carry adders are widely studied due to their straightforward design and low gate count [29, 77–81]. The main drawback of this approach is the linear depth of all the circuits proposed using this method. Despite this limitation, ripple-carry adders remain attractive due to their low qubit overhead and ease of implementation based on reversible classical gates such as the $\{X, \text{CNOT, Toffoli}\}$ gate set. Carry-lookahead adders, on the other hand, offer improved depth by precomputing carry signals [82–84]. However, this improvement in depth comes at the cost of requiring additional ancillary qubits, which can be a significant drawback in the case where qubit resources are limited. Another approach leverages the Quantum Fourier Transform (QFT) to perform addition [85]. This QFT-based adder can achieve logarithmic depth, but only in the case where the addition is approximated. Moreover, this approach relies on small-angle rotation gates, which are expensive to implement fault-tolerantly. A summary of the costs of the different approaches is given in Table 4.1.

In this chapter, we present a quantum ripple-carry adder that achieves a polylogarithmic depth of $\mathcal{O}(\log_2^2 n)$ and does not rely on any ancillary qubits. This is the first exact and ancilla-free quantum adder with sublinear depth. Our design bridges the gap between the ancilla-free approach of ripple-carry adders and the improved depth of carry-lookahead adders, offering a particularly low space-time cost.

In Section 4.1, we propose an algorithm for implementing the CNOT ladder operator with a logarithmic depth and a linear number of CNOT gates. Then, in Section 4.2, we extend this result for the implementation of the Toffoli ladder operator with a polylogarithmic depth and a linearithmic number of $\{\text{Toffoli}, X\}$ gates. Based on theses results, we present a polylogarithmic-depth and ancilla-free adder in Section 4.3, with a linearithmic number of $\{\text{Toffoli}, \text{CNOT}, X\}$ gates. Finally, in Section 4.4, we extend this result for the controlled addition, with the same asymptotic depth and gate count complexities. In the case where the addition is controlled by multiple qubits, we show that the depth complexity scales logarithmically with the number of controlled qubits.

Adder	Ancillae	Depth	Gate count
Carry-lookahead [82, 84]	$\mathcal{O}(n)$	$\mathcal{O}(\log_2(n))$	$\mathcal{O}(n)$
Carry-lookahead [29, 83]	$\mathcal{O}\left(\frac{n}{\log_2(n)}\right)$	$\mathcal{O}(\log_2(n))$	$\mathcal{O}(n)$
QFT-based (approx.) [85]	0	$\mathcal{O}(\log_2(n))$	$\mathcal{O}(n\log_2(n))$
QFT-based (exact) [85]	0	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$
Ripple-carry [78]	1	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Ripple-carry [29, 79]	0	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Proposed	0	$\mathcal{O}(\log_2^2(n))$	$\mathcal{O}(n\log_2(n))$

Table 4.1: Comparison of state-of-the-art quantum adders in terms of ancillae, depth, and gate count.

4.1 Logarithmic-depth implementation of the CNOT ladder operator

In this section, we present a logarithmic-depth implementation of the ladder operator, introduced in Section 1.2.3.3 and denoted by $\mathcal{L}_1^{(n-1)}$ over n qubits. Consider the pseudocode presented in Algorithm 1, in which :: denotes the concatenation operator. The algorithm constructs two CNOT circuits of depth 1, denoted by C_L and C_R, and performs a recursive call on $\lfloor n/2 \rfloor$ qubits (denoted by X') to produce a subcircuit that is inserted between C_L and C_R. For illustration, Figure 4.1 provides an example of the CNOT circuit resulting from this procedure when $n = 10$. The exact CNOT-depth and CNOT-count of the circuit produced by Algorithm 1 are stated in the following theorem.

Theorem 1. *The circuit produced by Algorithm 1 implements the $\mathcal{L}_1^{(n-1)}$ operator, where $n \geq 2$, with a CNOT-depth of*

$$\lfloor \log_2(n) \rfloor + \left\lfloor \log_2\left(\frac{2n}{3}\right) \right\rfloor, \tag{4.1}$$

and a CNOT-count of

$$2n - 2 - \lfloor \log_2(n) \rfloor - \left\lfloor \log_2\left(\frac{2n}{3}\right) \right\rfloor. \tag{4.2}$$

Proof. We first prove that the circuit produced by Algorithm 1 implements $\mathcal{L}_1^{(n-1)}$. For the base case where $n = 1$, $\mathcal{L}_1^{(0)}$ is equal to the identity operator, which corresponds to the empty circuit returned by Algorithm 1. For the other base case where $n = 2$, the algorithm produces a circuit containing a single CNOT gate, which corresponds to the implementation of the $\mathcal{L}_1^{(1)}$ operator:

$$\text{CNOT} \left|x_0\right\rangle \left|x_1\right\rangle = \left|x_0\right\rangle \left|x_0 \oplus x_1\right\rangle = \mathcal{L}_1^{(1)} \left|x_0\right\rangle \left|x_1\right\rangle. \tag{4.3}$$

For the other cases where $n > 2$, Algorithm 1 constructs two circuits, C_L and C_R, and performs a recursive call with a subset of qubits X', which produces a circuit that we denote by $C_{X'}$. The circuit produced by Algorithm 1 is then the result of the concatenation of the circuits C_L, $C_{X'}$, and C_R. Let U_L, $U_{X'}$, and U_R be the unitary operators associated with the circuits C_L, $C_{X'}$, and C_R, respectively. The action of the U_L operator on an n-dimensional computational basis

Algorithm 1: Logarithmic-depth CNOT circuit synthesis for $\mathcal{L}_1^{(n-1)}$

Input: A list $X = X_0, \ldots, X_{n-1}$ of n qubits.

Output: A circuit implementing the $\mathcal{L}_1^{(n-1)}$ operator over the qubits X.

procedure `Ladder`$_1$`-Synth`(X)
  if $n = 1$ **then**
    return empty circuit
  end
  if $n = 2$ **then**
    return $\mathrm{CNOT}(X_0, X_1)$
  end
  $X' \leftarrow X_1$
  $C_R \leftarrow \mathrm{CNOT}(X_0, X_1)$
  $C_L \leftarrow \mathrm{CNOT}(X_{n-2}, X_{n-1})$
  for $i = 1$ to $\lceil n/2 \rceil - 2$ **do**
    $C_L \leftarrow C_L :: \mathrm{CNOT}(X_{2i-1}, X_{2i})$
    $C_R \leftarrow C_R :: \mathrm{CNOT}(X_{2i}, X_{2i+1})$
    $X' \leftarrow X' :: X_{2i+1}$
  end
  if n is even **then**
    $X' \leftarrow X' :: X_{n-2}$
  end
  return $C_L ::$ `Ladder`$_1$`-Synth`$(X') :: C_R$

state $|\boldsymbol{x}\rangle$ is as follows:

$$U_L |\boldsymbol{x}\rangle = |x_0\rangle |x_1\rangle \left(\bigotimes_{i=1}^{\lceil \frac{n}{2} \rceil - 2} |x_{2i-1} \oplus x_{2i}\rangle |x_{2i+1}\rangle \right) \left(\bigotimes_{n \bmod 2}^{0} |x_{n-2}\rangle \right) |x_{n-2} \oplus x_{n-1}\rangle$$

The operator $U_{X'}$ implements the $\mathcal{L}_1^{(\lfloor n/2 \rfloor - 1)}$ operator on the qubits X', which corresponds to the sequence of qubits X_{2i+1} for all i satisfying $0 \leq i \leq \lceil \frac{n}{2} \rceil - 2$, followed by X_{n-2} when n is even. Thus, the action of the $U_{X'}$ operator on an n-dimensional computational basis state $|\boldsymbol{x}\rangle$ is as follows:

$$U_{X'} |\boldsymbol{x}\rangle = |x_0\rangle |x_1\rangle \left(\bigotimes_{i=1}^{\lceil \frac{n}{2} \rceil - 2} |x_{2i}\rangle |x_{2i-1} \oplus x_{2i+1}\rangle \right) \left(\bigotimes_{n \bmod 2}^{0} |x_{n-3} \oplus x_{n-2}\rangle \right) |x_{n-1}\rangle$$

And the action of the U_R operator on an n-dimensional computational basis state $|\boldsymbol{x}\rangle$ is as follows:

$$U_R |\boldsymbol{x}\rangle = |x_0\rangle |x_0 \oplus x_1\rangle \left(\bigotimes_{i=1}^{\lceil \frac{n}{2} \rceil - 2} |x_{2i}\rangle |x_{2i} \oplus x_{2i+1}\rangle \right) \left(\bigotimes_{n \bmod 2}^{0} |x_{n-2}\rangle \right) |x_{n-1}\rangle$$

By putting these equations together, the operation performed by the $U_R U_{X'} U_L$ operator can be

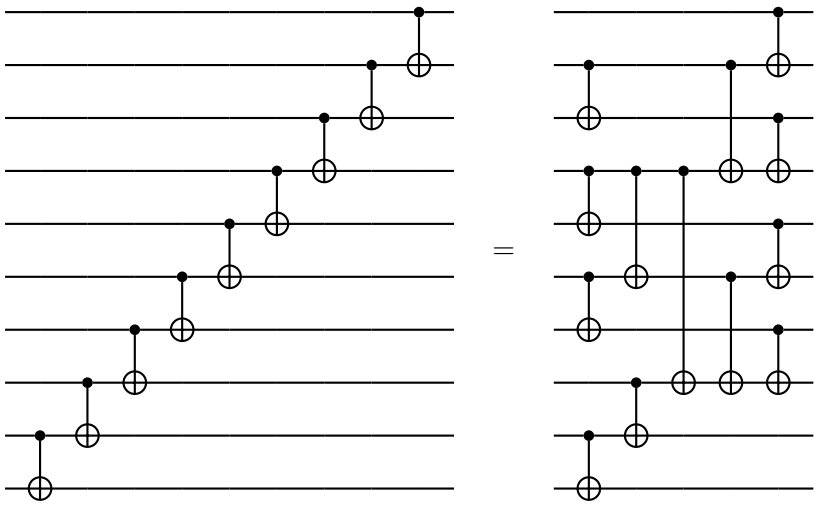


Figure 4.1: On the left, a linear-depth circuit implementing the $\mathcal{L}_1^{(9)}$ operator. On the right, an equivalent logarithmic-depth circuit produced by Algorithm 1.

described as follows:

$$
\begin{aligned}
&U_R U_{X'} U_L \left|\boldsymbol{x}\right\rangle \\
&= U_R U_{X'} \left[\left|x_0\right\rangle \left|x_1\right\rangle \left(\bigotimes_{i=1}^{\lceil \frac{n}{2} \rceil - 2} \left|x_{2i-1} \oplus x_{2i}\right\rangle \left|x_{2i+1}\right\rangle \right) \left(\bigotimes_{n \bmod 2}^{0} \left|x_{n-2}\right\rangle \right) \left|x_{n-2} \oplus x_{n-1}\right\rangle \right] \\
&= U_R \left[\left|x_0\right\rangle \left|x_1\right\rangle \left(\bigotimes_{i=1}^{\lceil \frac{n}{2} \rceil - 2} \left|x_{2i-1} \oplus x_{2i}\right\rangle \left|x_{2i-1} \oplus x_{2i+1}\right\rangle \right) \left(\bigotimes_{n \bmod 2}^{0} \left|x_{n-3} \oplus x_{n-2}\right\rangle \right) \left|x_{n-2} \oplus x_{n-1}\right\rangle \right] \\
&= \left|x_0\right\rangle \left|x_0 \oplus x_1\right\rangle \left(\bigotimes_{i=1}^{\lceil \frac{n}{2} \rceil - 2} \left|x_{2i-1} \oplus x_{2i}\right\rangle \left|x_{2i} \oplus x_{2i+1}\right\rangle \right) \left(\bigotimes_{n \bmod 2}^{0} \left|x_{n-3} \oplus x_{n-2}\right\rangle \right) \left|x_{n-2} \oplus x_{n-1}\right\rangle \\
&= \left|x_0\right\rangle \bigotimes_{i=1}^{n-1} \left|x_i \oplus x_{i-1}\right\rangle \\
&= \mathcal{L}_1^{(n-1)} \left|\boldsymbol{x}\right\rangle
\end{aligned}
$$

Thus, the circuit produced by Algorithm 1, associated with the operator $U_R U_{X'} U_L$, produces a circuit implementing the $\mathcal{L}_1^{(n-1)}$ operator.

The CNOT-depth of the U_L and U_R circuits is exactly one, because all the CNOT gates in the circuits C_L and C_R are applied on different qubits. Therefore, the depth $D(n)$ of the circuit produced by Algorithm 1 is

$$D(n) = 2 + D(\lfloor n/2 \rfloor) \tag{4.4}$$

with $D(1) = 0$ and $D(2) = 1$. For $n > 2$, the algorithm performs $\lfloor \log_2(n) \rfloor$ recursive calls (including the initial function call), plus an additional one when n satisfies

$$n \geq \frac{2^{\lfloor \log_2(n) \rfloor} + 2^{\lfloor \log_2(n) \rfloor + 1}}{2} = \frac{3}{2} 2^{\lfloor \log_2(n) \rfloor} \tag{4.5}$$

which has a CNOT-depth cost of 0 (corresponding to the base case where $n = 1$). We then have:

$$\begin{aligned} & n \geq \frac{3}{2} 2^{\lfloor \log_2(n) \rfloor} \\ \implies & \log_2\left(\frac{2n}{3}\right) \geq \lfloor \log_2(n) \rfloor \\ \implies & \log_2\left(\frac{2n}{3}\right) - \lfloor \log_2(n) \rfloor \geq 0 \\ \implies & \left\lfloor \log_2\left(\frac{2n}{3}\right) \right\rfloor - \lfloor \log_2(n) \rfloor = 0. \end{aligned} \tag{4.6}$$

Thus, the value of

$$\left\lfloor \log_2\left(\frac{2n}{3}\right) \right\rfloor - \lfloor \log_2(n) \rfloor \tag{4.7}$$

is equal to 0 if and only if n satisfies Equation 4.5, or -1 otherwise. The number of recursive calls (including the initial function call) is therefore equal to

$$\lfloor \log_2(n) \rfloor + \left\lfloor \log_2\left(\frac{2n}{3}\right) \right\rfloor - \lfloor \log_2(n) \rfloor + 1 = \left\lfloor \log_2\left(\frac{2n}{3}\right) \right\rfloor + 1. \tag{4.8}$$

Each recursive call induces a CNOT-depth cost of 2 from the circuits C_L and C_R, except the last call, which has a cost of 0 if Equation 4.5 is satisfied, and 1 otherwise. For $n \geq 2$, the total CNOT-depth of the circuit produced by Algorithm 1 is then equal to

$$\begin{aligned} D(n) &= 2 \left\lfloor \log_2\left(\frac{2n}{3}\right) \right\rfloor - \left(\left\lfloor \log_2\left(\frac{2n}{3}\right) \right\rfloor - \lfloor \log_2(n) \rfloor \right) \\ &= \lfloor \log_2(n) \rfloor + \left\lfloor \log_2\left(\frac{2n}{3}\right) \right\rfloor. \end{aligned} \tag{4.9}$$

The number of CNOT gates in the C_L and C_R circuits is

$$\left\lfloor \frac{n-1}{2} \right\rfloor, \tag{4.10}$$

which implies that the number of CNOT gates in the circuit produced by Algorithm 1 is

$$C(n) = 2 \left\lfloor \frac{n-1}{2} \right\rfloor + C\left(\left\lfloor \frac{n}{2} \right\rfloor \right) \tag{4.11}$$

with $C(2) = 1$ and $C(3) = 2$. In the case where $n = 2^k$, we have:

$$\begin{aligned} C(n) &= n - 2 + C(n/2) \\ &= \sum_{i=0}^{\log_2(n)-2} \left(\frac{n}{2^i} - 2 \right) + C(2) \\ &= \sum_{i=0}^{\log_2(n)-2} \frac{n}{2^i} - 2(\log_2(n) - 1) + 1 \\ &= 2n - 4 - 2\log_2(n) + 3 \\ &= 2n - 2 - D(n). \end{aligned} \tag{4.12}$$

Similarly, in the case where $n = 3 \times 2^k$, we have:

$$\begin{aligned} C(n) &= n - 2 + C(n/2) \\ &= \sum_{i=0}^{\lfloor \log_2(n) \rfloor - 2} \left(\frac{n}{2^i} - 2 \right) + C(3) \\ &= \sum_{i=0}^{\lfloor \log_2(n) \rfloor - 2} \frac{n}{2^i} - 2(\lfloor \log_2(n) \rfloor - 1) + 2 \\ &= 2n - 6 - 2 \lfloor \log_2(n) \rfloor + 4 \\ &= 2n - 2 - D(n). \end{aligned} \tag{4.13}$$

Moreover, in the case where n is odd, we have:

$$\begin{aligned} C(n) &= 2 \left\lfloor \frac{n-1}{2} \right\rfloor + C(\lfloor n/2 \rfloor) \\ &= 2 + 2 \left\lfloor \frac{n-2}{2} \right\rfloor + C(\lfloor (n-1)/2 \rfloor) \\ &= 2 + C(n-1). \end{aligned} \tag{4.14}$$

Note that this equality implies that

$$C(n) - C(n-1) = 2 \tag{4.15}$$

when n is odd. Finally, in the case where n is even, $n \neq 2^k$ and $n \neq 3 \times 2^k$, we have:

$$\begin{aligned} C(n) &= 2 \left\lfloor \frac{n-1}{2} \right\rfloor + C(\lfloor n/2 \rfloor) \\ &= 2 \left\lfloor \frac{n-2}{2} \right\rfloor + C(\lfloor (n-1)/2 \rfloor + 1) \\ &= C(n-1) - C(\lfloor (n-1)/2 \rfloor) + C(\lfloor n/2 \rfloor) \\ &= C(n-1) - C(n/2 - 1) + C(n/2). \end{aligned} \tag{4.16}$$

In the case where $n/2$ is even, by applying the equality recursively, we obtain:

$$\begin{aligned} C(n) &= C(n-1) - C(n/2-1) + C(n/2) \\ &= C(n-1) - C(n/2-1) + C(n/2-1) - C(n/4-1) + C(n/4) \\ &= C(n-1) - C(n/4-1) + C(n/4). \end{aligned} \tag{4.17}$$

Thus, based on Equation 4.15, we have the following equality, where k is the highest integer for which $n/2^k$ is odd:

$$\begin{aligned} C(n) &= C(n-1) - C(n/2-1) + C(n/2) \\ &= C(n-1) - C(n/2^k-1) + C(n/2^k) \\ &= 2 + C(n-1). \end{aligned} \tag{4.18}$$

Let $m = \max(2^{\lfloor \log_2(n) \rfloor}, 3 \times 2^{\lfloor \log_2(n/3) \rfloor})$, i.e. the closest value lower than n that is equal to 2^k or $3 \times 2^{k+1}$ for some integer k. Then, in the case where $n \neq 2^k$ and $n \neq 3 \times 2^k$, because $D(n) = D(n-1)$ and $C(n) = 2 + C(n-1)$, we have:

$$\begin{aligned} C(n) &= 2(n-m) + C(m) \\ &= 2(n-m) + 2m - 2 - D(n) \\ &= 2n - 2 - D(n). \end{aligned} \tag{4.19}$$

For any $n \geq 2$, the total CNOT-count of the circuit produced by Algorithm 1 is then equal to

$$\begin{aligned} C(n) &= 2n - 2 - D(n) \\ &= 2n - 2 - \lfloor \log_2(n) \rfloor - \left\lfloor \log_2\left(\frac{2n}{3}\right) \right\rfloor . \end{aligned} \tag{4.20}$$

□

4.2 Polylogarithmic-depth implementation of the Toffoli ladder operator

In this section, we present a polylogarithmic-depth implementation of the Toffoli ladder operator, introduced in Section 1.2.3.3 and denoted by $\mathcal{L}_2^{(n)}$ over $2n+1$ qubits.

We begin by presenting an algorithm for implementing the $\mathcal{L}_{\boldsymbol{\alpha}}$ operator (also introduced in Section 1.2.3.3) with logarithmic depth using MCX gates. Consider the pseudocode presented in Algorithm 2, where :: denotes the concatenation operator. Analogously to Algorithm 1, the algorithm constructs two MCX circuits of depth 1, denoted by C_L and C_R, and performs a recursive call to produce a subcircuit that is inserted between C_L and C_R. For illustration, Figure 4.2 provides an example of the CNOT circuit resulting from this procedure when $\boldsymbol{\alpha} = (2, 4, \ldots, 18)$. The exact MCX-depth and MCX-count of the circuit produced by Algorithm 1 are stated in the following theorem.

Theorem 2. *Let $\boldsymbol{\alpha}$ be a vector of $k-1$ integers associated with the $\mathcal{L}_{\boldsymbol{\alpha}}$ operator. The circuit produced by Algorithm 2 implements $\mathcal{L}_{\boldsymbol{\alpha}}$ with an MCX-depth of*

$$\lfloor \log_2(k) \rfloor + \left\lfloor \log_2\left(\frac{2k}{3}\right) \right\rfloor , \tag{4.21}$$

and an MCX-count of

$$2k - 2 - \lfloor \log_2(k) \rfloor - \left\lfloor \log_2\left(\frac{2k}{3}\right) \right\rfloor . \tag{4.22}$$

Proof. We first prove that the circuit produced by Algorithm 2 implements $\mathcal{L}_{\boldsymbol{\alpha}}$. For the base case where $k = 1$ (meaning that $\boldsymbol{\alpha}$ is empty), $\mathcal{L}_{\boldsymbol{\alpha}}$ is equal to the identity operator, which corresponds to the empty circuit returned by Algorithm 2. For the other base case where $k = 2$, the algorithm produces a circuit containing a single MCX gate applied on the qubits $(X_0, \ldots, X_{\alpha_0})$, which corresponds to the implementation of the $\mathcal{L}_{\boldsymbol{\alpha}}$ operator:

$$\begin{aligned} \mathrm{MCX}\left(\bigotimes_{i=0}^{\alpha_0} |x_i\rangle\right) &= \left(\bigotimes_{i=0}^{\alpha_0 - 1} |x_i\rangle\right) |x_{\alpha_0} \oplus \prod_{i=0}^{\alpha_0 - 1} x_i\rangle \\ &= \mathcal{L}_{\boldsymbol{\alpha}}\left(\bigotimes_{i=0}^{\alpha_0} |x_i\rangle\right) . \end{aligned} \tag{4.23}$$

For the other cases where $k > 2$, Algorithm 2 constructs two circuits, C_L and C_R, and performs a recursive call with parameters $\boldsymbol{\alpha}'$ and a subset of qubits X', which produces a circuit that we denote by $C_{X'}$. The circuit produced by Algorithm 2 is then the result of the concatenation of the circuits C_L, $C_{X'}$, and C_R. Let U_L, $U_{X'}$, and U_R be the unitary operators associated with

Algorithm 2: Logarithmic-depth MCX circuit synthesis for $\mathcal{L}_{\boldsymbol{\alpha}}$

Input: A vector $\boldsymbol{\alpha}$ of $k-1$ integers associated with the $\mathcal{L}_{\boldsymbol{\alpha}}$ operator, and a list $X = X_0, \ldots, X_{\alpha_{k-2}}$ of $\alpha_{k-2}+1$ qubits.

Output: A circuit implementing the $\mathcal{L}_{\boldsymbol{\alpha}}$ operator over the qubits X.

procedure `Ladder`$_{\boldsymbol{\alpha}}$`-Synth`(X, $\boldsymbol{\alpha}$)
  if $k = 1$ **then**
    return empty circuit
  end
  if $k = 2$ **then**
    return $\mathrm{MCX}(X_0, \ldots, X_{\alpha_0})$
  end
  $X' \leftarrow$ empty list of qubits
  $\boldsymbol{\alpha}' \leftarrow$ empty vector of integers
  $C_R \leftarrow \mathrm{MCX}(X_0, \ldots, X_{\alpha_0})$
  $C_L \leftarrow \mathrm{MCX}(X_{\alpha_{k-3}}, \ldots, X_{\alpha_{k-2}})$
  for $i = 1$ to $\lceil k/2 \rceil - 2$ **do**
    $C_L \leftarrow C_L :: \mathrm{MCX}(X_{\alpha_{2i-2}}, \ldots, X_{\alpha_{2i-1}})$
    $C_R \leftarrow C_R :: \mathrm{MCX}(X_{\alpha_{2i-1}}, \ldots, X_{\alpha_{2i}})$
    $X' \leftarrow X' :: \left[X_{\alpha_{2i-2}+1}, \ldots, X_{\alpha_{2i-1}-1}, X_{\alpha_{2i-1}+1}, \ldots, X_{\alpha_{2i}}\right]$
    $\boldsymbol{\alpha}' \leftarrow \boldsymbol{\alpha}' :: \alpha_{2i} - \alpha_0 - i$
  end
  if k is even **then**
    $X' \leftarrow X' :: \left[X_{\alpha_{k-4}+1}, \ldots, X_{\alpha_{k-3}}\right]$
    $\boldsymbol{\alpha}' \leftarrow \boldsymbol{\alpha}' :: \alpha_{k-3} - \alpha_0 - k/2 - 2$
  end
  return $C_L ::$ `Ladder`$_{\alpha}$`-Synth`(X', $\boldsymbol{\alpha}'$) $:: C_R$

the circuits C_L, $C_{X'}$, and C_R, respectively. The action of the U_L operator on an $(\alpha_{k-2}+1)$-dimensional computational basis state $|\boldsymbol{x}\rangle$ is as follows:

$$U_L |\boldsymbol{x}\rangle = \bigotimes_{i=0}^{\alpha_0} |x_i\rangle \left(\bigotimes_{i=1}^{\lceil \frac{k}{2} \rceil - 2} \left(\bigotimes_{j=\alpha_{2i-2}+1}^{\alpha_{2i-1}-1} |x_j\rangle \right) |x_{\alpha_{2i-1}} \oplus \prod_{j=\alpha_{2i-2}}^{\alpha_{2i-1}-1} x_j\rangle \bigotimes_{j=\alpha_{2i-1}+1}^{\alpha_{2i}} |x_j\rangle \right)$$
$$\left(\bigotimes_{k \bmod 2}^{0} \bigotimes_{i=\alpha_{k-4}+1}^{\alpha_{k-3}} |x_i\rangle \right) \left(\bigotimes_{i=\alpha_{k-3}+1}^{\alpha_{k-2}-1} |x_i\rangle \right) |x_{\alpha_{k-2}} \oplus \prod_{j=\alpha_{k-3}}^{\alpha_{k-2}-1} x_j\rangle .$$

The operator $U_{X'}$ implements the $\mathcal{L}_{\boldsymbol{\alpha}'}$ operator on the $\alpha'_{\lfloor k/2 \rfloor - 2} + 1$ qubits X'. Thus, the action of the $U_{X'}$ operator on an $(\alpha_{k-2}+1)$-dimensional computational basis state $|\boldsymbol{x}\rangle$ is as follows:

$$U_{X'} |\boldsymbol{x}\rangle = \bigotimes_{i=0}^{\alpha_0} |x_i\rangle \left(\bigotimes_{i=1}^{\lceil \frac{k}{2} \rceil - 2} \left(\bigotimes_{j=\alpha_{2i-2}+1}^{\alpha_{2i-1}} |x_j\rangle \right) \left(\bigotimes_{j=\alpha_{2i-1}+1}^{\alpha_{2i}-1} |x_j\rangle \right) |x_{\alpha_{2i}} \oplus \prod_{\substack{j=\alpha_{2i-2} \\ j \neq \alpha_{2i-1}}}^{\alpha_{2i}-1} x_j\rangle \right)$$
$$\left(\bigotimes_{k \bmod 2}^{0} \left(\bigotimes_{i=\alpha_{k-4}+1}^{\alpha_{k-3}-1} |x_i\rangle \right) |x_{\alpha_{k-3}} \oplus \prod_{j=\alpha_{k-4}}^{\alpha_{k-3}-1} x_j\rangle \right) \bigotimes_{i=\alpha_{k-3}+1}^{\alpha_{k-2}} |x_i\rangle .$$

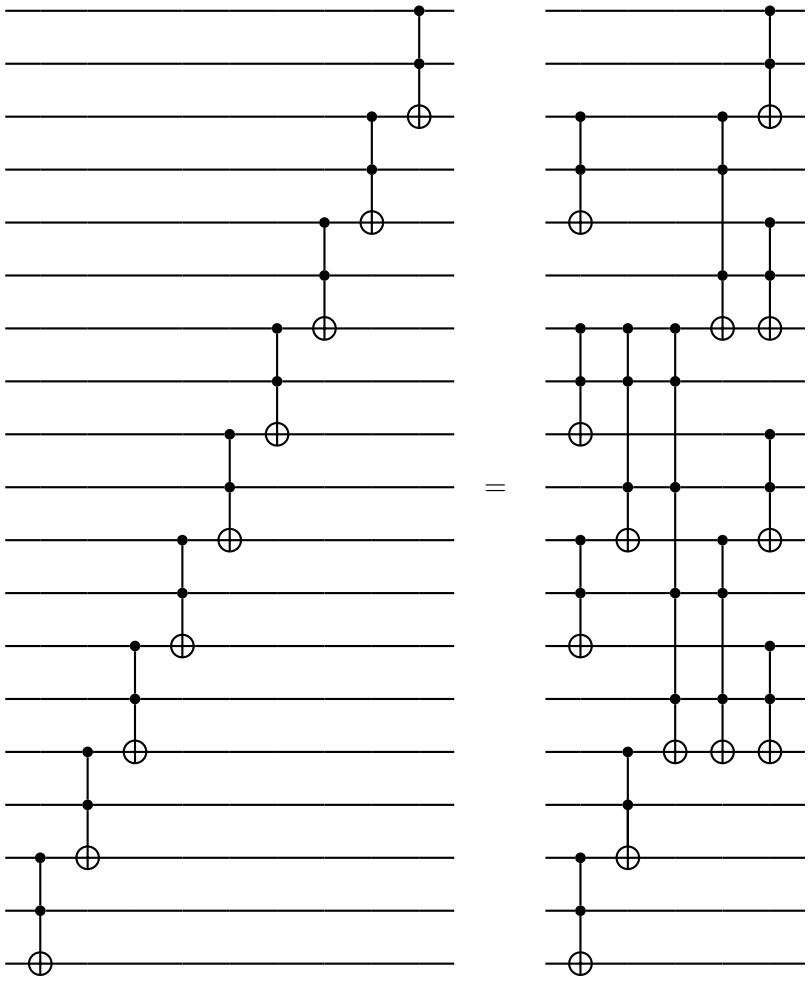


Figure 4.2: On the left, a linear-depth circuit implementing the $\mathcal{L}_2^{(9)}$ operator. On the right, an equivalent logarithmic-depth circuit produced by Algorithm 2.

And the action of the U_R operator on an $(\alpha_{k-2}+1)$-dimensional computational basis state $|\boldsymbol{x}\rangle$ is as follows:

$$U_R|\boldsymbol{x}\rangle = \left(\bigotimes_{i=0}^{\alpha_0-1}|x_i\rangle\,|x_{\alpha_0}\oplus\prod_{i=0}^{\alpha_0-1}x_i\rangle\right)\left(\bigotimes_{i=1}^{\lceil\frac{k}{2}\rceil-2}\left(\bigotimes_{j=\alpha_{2i-2}+1}^{\alpha_{2i-1}}|x_j\rangle\right)\left(\bigotimes_{j=\alpha_{2i-1}+1}^{\alpha_{2i}-1}|x_j\rangle\right)|x_{\alpha_{2i}}\oplus\prod_{j=\alpha_{2i-1}}^{\alpha_{2i}-1}x_j\rangle\right)$$
$$\left(\bigotimes_{k \bmod 2}\bigotimes_{i=\alpha_{k-4}+1}^{\alpha_{k-3}}|x_i\rangle\right)\bigotimes_{i=\alpha_{k-3}+1}^{\alpha_{k-2}}|x_i\rangle\,.$$

By putting these equations together, the operation performed by the $U_R U_{X'} U_L$ operator can be

described as follows:

$$
\begin{aligned}
&U_R U_{X'} U_L \left|\boldsymbol{x}\right\rangle \\
&= U_R U_{X'} \left[\bigotimes_{i=0}^{\alpha_0} \left|x_i\right\rangle \left(\bigotimes_{i=1}^{\lceil \frac{k}{2} \rceil - 2} \left(\bigotimes_{j=\alpha_{2i-2}+1}^{\alpha_{2i-1}-1} \left|x_j\right\rangle \right) \left|x_{\alpha_{2i-1}} \oplus \prod_{j=\alpha_{2i-2}}^{\alpha_{2i-1}-1} x_j\right\rangle \bigotimes_{j=\alpha_{2i-1}+1}^{\alpha_{2i}} \left|x_j\right\rangle \right) \right. \\
&\quad \left. \left(\bigotimes_{k \bmod 2}^{0} \bigotimes_{i=\alpha_{k-4}+1}^{\alpha_{k-3}} \left|x_i\right\rangle \right) \left(\bigotimes_{i=\alpha_{k-3}+1}^{\alpha_{k-2}-1} \left|x_i\right\rangle \right) \left|x_{\alpha_{k-2}} \oplus \prod_{j=\alpha_{k-3}}^{\alpha_{k-2}-1} x_j\right\rangle \right] \\
&= U_R \left[\bigotimes_{i=0}^{\alpha_0} \left|x_i\right\rangle \left(\bigotimes_{i=1}^{\lceil \frac{k}{2} \rceil - 2} \left(\bigotimes_{j=\alpha_{2i-2}+1}^{\alpha_{2i-1}-1} \left|x_j\right\rangle \right) \left|x_{\alpha_{2i-1}} \oplus \prod_{j=\alpha_{2i-2}}^{\alpha_{2i-1}-1} x_j\right\rangle \right. \right. \\
&\quad \left. \left(\bigotimes_{j=\alpha_{2i-1}+1}^{\alpha_{2i}-1} \left|x_j\right\rangle \right) \left|x_{\alpha_{2i}} \oplus \prod_{\substack{j=\alpha_{2i-2} \\ j \neq \alpha_{2i-1}}}^{\alpha_{2i}-1} x_j\right\rangle \right) \\
&\quad \left. \left(\bigotimes_{k \bmod 2}^{0} \left(\bigotimes_{i=\alpha_{k-4}+1}^{\alpha_{k-3}-1} \left|x_i\right\rangle \right) \left|x_{\alpha_{k-3}} \oplus \prod_{j=\alpha_{k-4}}^{\alpha_{k-3}-1} x_j\right\rangle \right) \left(\bigotimes_{i=\alpha_{k-3}+1}^{\alpha_{k-2}-1} \left|x_i\right\rangle \right) \left|x_{\alpha_{k-2}} \oplus \prod_{j=\alpha_{k-3}}^{\alpha_{k-2}-1} x_j\right\rangle \right] \\
&= \left(\bigotimes_{i=0}^{\alpha_0-1} \left|x_i\right\rangle \left|x_{\alpha_0} \oplus \prod_{j=0}^{\alpha_0-1} x_j\right\rangle \right) \left(\bigotimes_{i=1}^{\lceil \frac{k}{2} \rceil - 2} \left(\bigotimes_{j=\alpha_{2i-2}+1}^{\alpha_{2i-1}-1} \left|x_j\right\rangle \right) \left|x_{\alpha_{2i-1}} \oplus \prod_{j=\alpha_{2i-2}}^{\alpha_{2i-1}-1} x_j\right\rangle \right. \\
&\quad \left. \left(\bigotimes_{j=\alpha_{2i-1}+1}^{\alpha_{2i}-1} \left|x_j\right\rangle \right) \left|x_{\alpha_{2i}} \oplus \prod_{\substack{j=\alpha_{2i-2} \\ j \neq \alpha_{2i-1}}}^{\alpha_{2i}-1} x_j \oplus \left(x_{\alpha_{2i-1}} \oplus \prod_{j=\alpha_{2i-2}}^{\alpha_{2i-1}-1} x_j \right) \prod_{\alpha_{2i-1}+1}^{\alpha_{2i}-1} x_j\right\rangle \right) \\
&\quad \left(\bigotimes_{k \bmod 2}^{0} \left(\bigotimes_{i=\alpha_{k-4}+1}^{\alpha_{k-3}-1} \left|x_i\right\rangle \right) \left|x_{\alpha_{k-3}} \oplus \prod_{j=\alpha_{k-4}}^{\alpha_{k-3}-1} x_j\right\rangle \right) \left(\bigotimes_{i=\alpha_{k-3}+1}^{\alpha_{k-2}-1} \left|x_i\right\rangle \right) \left|x_{\alpha_{k-2}} \oplus \prod_{j=\alpha_{k-3}}^{\alpha_{k-2}-1} x_j\right\rangle \\
&= \left(\bigotimes_{i=0}^{\alpha_0-1} \left|x_i\right\rangle \left|x_{\alpha_0} \oplus \prod_{j=0}^{\alpha_0-1} x_j\right\rangle \right) \left(\bigotimes_{i=1}^{\lceil \frac{k}{2} \rceil - 2} \left(\bigotimes_{j=\alpha_{2i-2}+1}^{\alpha_{2i-1}-1} \left|x_j\right\rangle \right) \left|x_{\alpha_{2i-1}} \oplus \prod_{j=\alpha_{2i-2}}^{\alpha_{2i-1}-1} x_j\right\rangle \right. \\
&\quad \left. \left(\bigotimes_{j=\alpha_{2i-1}+1}^{\alpha_{2i}-1} \left|x_j\right\rangle \right) \left|x_{\alpha_{2i}} \oplus \prod_{\alpha_{2i-1}}^{\alpha_{2i}-1} x_j\right\rangle \right) \\
&\quad \left(\bigotimes_{k \bmod 2}^{0} \left(\bigotimes_{i=\alpha_{k-4}+1}^{\alpha_{k-3}-1} \left|x_i\right\rangle \right) \left|x_{\alpha_{k-3}} \oplus \prod_{j=\alpha_{k-4}}^{\alpha_{k-3}-1} x_j\right\rangle \right) \left(\bigotimes_{i=\alpha_{k-3}+1}^{\alpha_{k-2}-1} \left|x_i\right\rangle \right) \left|x_{\alpha_{k-2}} \oplus \prod_{j=\alpha_{k-3}}^{\alpha_{k-2}-1} x_j\right\rangle \\
&= \left(\bigotimes_{i=0}^{\alpha_0-1} \left|x_i\right\rangle \left|x_{\alpha_0} \oplus \prod_{j=0}^{\alpha_0-1} x_j\right\rangle \right) \left(\bigotimes_{i=1}^{k-2} \left(\bigotimes_{j=\alpha_{i-1}+1}^{\alpha_i-1} \left|x_j\right\rangle \right) \left|x_{\alpha_i} \oplus \prod_{j=\alpha_{i-1}}^{\alpha_i-1} x_j\right\rangle \right) \\
&= \mathcal{L}_{\boldsymbol{\alpha}} \left|\boldsymbol{x}\right\rangle
\end{aligned}
$$

Thus, the circuit produced by Algorithm 2, associated with the operator $U_R U_{X'} U_L$, produces a

circuit implementing the $\mathcal{L}_{\boldsymbol{\alpha}}$ operator.

The MCX-depth of the C_L and C_R circuits is exactly one, because all the MCX gates in these circuits are applied on different qubits. Therefore, the depth $D(k)$ of the circuit produced by Algorithm 2 is

$$D(k) = 2 + D\left(\left\lfloor \frac{k}{2} \right\rfloor\right) \tag{4.24}$$

with $D(1) = 0$ and $D(2) = 1$. This equation is equivalent to Equation 4.4, which was demonstrated in the proof of Theorem 1 to be equal to

$$D(k) = \lfloor \log_2(k) \rfloor + \left\lfloor \log_2\left(\frac{2k}{3}\right) \right\rfloor. \tag{4.25}$$

for $k \geq 2$.

Finally, the number of MCX gates in the C_L and C_R circuits is

$$\left\lfloor \frac{k-1}{2} \right\rfloor, \tag{4.26}$$

which implies that the number of MCX gates in the circuit produced by Algorithm 2 is

$$C(k) = 2\left\lfloor \frac{k-1}{2} \right\rfloor + C\left(\left\lfloor \frac{k}{2} \right\rfloor\right) \tag{4.27}$$

with $C(2) = 1$ and $C(3) = 2$. This equation is equivalent to Equation 4.11, which was demonstrated in the proof of Theorem 1 to be equal to

$$C(k) = 2k - 2 - \lfloor \log_2(k) \rfloor - \left\lfloor \log_2\left(\frac{2k}{3}\right) \right\rfloor \tag{4.28}$$

for $k \geq 2$. □

This theorem entails the following corollary for the circuit synthesis of the $\mathcal{L}_2^{(n)}$ operator using MCX gates.

Corollary 1. *Algorithm 2 can produce a circuit implementing the $\mathcal{L}_2^{(n-1)}$ operator, where $n \geq 2$, with an MCX-depth of*

$$\lfloor \log_2(n) \rfloor + \left\lfloor \log_2\left(\frac{2n}{3}\right) \right\rfloor, \tag{4.29}$$

and an MCX-count of

$$2n - 2 - \lfloor \log_2(n) \rfloor - \left\lfloor \log_2\left(\frac{2n}{3}\right) \right\rfloor. \tag{4.30}$$

Proof. Let $\boldsymbol{\alpha}$ be a vector of size $(n-1)$ such that $\boldsymbol{\alpha} = (2, 4, \ldots, 2(n-1))$, and let C be the circuit produced by Algorithm 2 when $\boldsymbol{\alpha}$ is given as input. We have

$$\begin{aligned}
\mathcal{L}_{\boldsymbol{\alpha}} |\boldsymbol{x}\rangle &= \left(\bigotimes_{i=0}^{\alpha_0-1} |x_i\rangle\right) |x_{\alpha_0} \oplus \prod_{j=0}^{\alpha_0-1} x_j\rangle \bigotimes_{i=1}^{n-2} \left(\bigotimes_{j=\alpha_{i-1}+1}^{\alpha_i-1} |x_j\rangle\right) |x_{\alpha_i} \oplus \prod_{j=\alpha_{i-1}}^{\alpha_i-1} x_j\rangle \\
&= |x_0\rangle \bigotimes_{i=0}^{n-2} |x_{2i+1}\rangle \, |x_{2i+2} \oplus x_{2i}x_{2i+1}\rangle \\
&= \mathcal{L}_2^{(n-1)} |\boldsymbol{x}\rangle
\end{aligned} \tag{4.31}$$

which implies that C implements the $\mathcal{L}_2^{(n)}$ operator. Moreover, as stated by Theorem 2, C implements $\mathcal{L}_{\boldsymbol{\alpha}}$ with an MCX-depth of

$$\lfloor \log_2(n) \rfloor + \left\lfloor \log_2\left(\frac{2n}{3}\right) \right\rfloor, \tag{4.32}$$

and an MCX-count of

$$2n - 2 - \lfloor \log_2(n) \rfloor - \left\lfloor \log_2\left(\frac{2n}{3}\right) \right\rfloor. \tag{4.33}$$

□

The logarithmic-depth MCX-circuit produced by Algorithm 2 can be translated into a {Toffoli, X} circuit by using the MCX gate implementation presented in Reference [26]. We obtain a polylogarithmic-depth circuit, as stated by the following theorem.

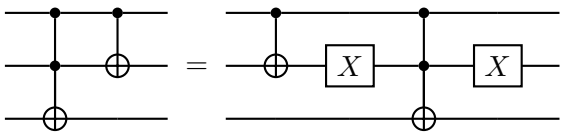
Theorem 3. *The $\mathcal{L}_2^{(n)}$ operator can be implemented over the $\{Toffoli, X\}$ gate set with a depth of $\mathcal{O}(\log_2^2(n))$ and a gate count of $\mathcal{O}(n \log_2(n))$, without any ancillary qubits.*

Proof. Consider the MCX circuit produced by Algorithm 2 when $\boldsymbol{\alpha} = (2, 4, \ldots, 2(n-1))$ is given as input. This circuit implements the $\mathcal{L}_2^{(n)}$ operator, as demonstrated in the proof of Corollary 1. The first and last layers of the circuit are composed of parallel Toffoli gates, inducing a depth of 2 and a number of Toffoli gates of $\mathcal{O}(n)$. The first two qubits of the circuit are not used in any other layer of the circuit. Moreover, for all the other layers of the circuit, the parallel MCX gates are all separated by at least two qubits on which no gates are acting. Therefore, for each one of these layers, two different dirty ancillary qubits can be associated to each MCX gate. Then, based on Reference [26], we can implement all the MCX gates in a given layer in parallel over the {Toffoli, X} gate set, with a depth of $\mathcal{O}(\log_2(m_i))$ and a gate count of $\mathcal{O}(m_i)$ for each gate, where m_i is the number of controls of the i-th MCX gate in the layer. We have $\max_i(m_i) \leq n$, which implies that the total depth of the layer is $\mathcal{O}(\log_2(n))$, as all the MCX gates are implemented in parallel. Moreover, we have $\sum_i m_i \leq n$, which implies that the total number of {Toffoli, X} gates in the layer is $\mathcal{O}(n)$. As stated by Theorem 2, there are $\mathcal{O}(\log_2(n))$ layers of parallel MCX gates in the initial circuit, which results in a {Toffoli, X} circuit with a depth complexity of $\mathcal{O}(\log_2^2(n))$ and a gate count of $\mathcal{O}(n \log_2(n))$. □

4.3 Application to quantum addition

We can now rely on the results established in the previous sections to introduce the main result of this chapter: a polylogarithmic-depth and ancilla-free quantum adder.

Consider the ripple-carry adder presented in Algorithm 3. This adder is derived from the one presented in Reference [29] by applying the following circuit equality on the gates in the Slices 4 of the adder:



(4.34)

An example of the circuit produced by this adder for $n = 5$ is provided in Figure 4.3, where a simple linear-depth implementation of the first-order and second-order ladder operators is used.

Algorithm 3: Ancilla-free ripple-carry adder

Input: Two n-qubit registers $|a\rangle_A$ and $|b\rangle_B$, and a single-qubit register $|z\rangle_Z$.
Output: A circuit implementing the in-place addition of a and b:
$|a\rangle_A |a+b \mod 2^n\rangle_B |z \oplus (a+b)_n\rangle_Z$.

for $i = 1$ to $n-1$ **do** // Slice 1
  CNOT(A_i, B_i)
end
Apply $\mathcal{L}_1^{(n-1)}$ on $(A_1, \ldots, A_{n-1}, Z)$ // Slice 2
Apply $\left(\mathcal{L}_2^{(n-1)}\right)^\dagger$ on $(A_0, B_0, \ldots, A_{n-2}, B_{n-2}, A_{n-1})$ // Slice 3
for $i = 0$ to $n-1$ **do** // Slice 4
  CNOT(A_i, B_i)
end
for $i = 0$ to $n-1$ **do** // Slice 5
  $X(B_i)$
end
Apply $\mathcal{L}_2^{(n)}$ on $(A_0, B_0, \ldots, A_{n-1}, B_{n-1}, Z)$
for $i = 0$ to $n-1$ **do**
  $X(B_i)$
end
Apply $\left(\mathcal{L}_1^{(n-2)}\right)^\dagger$ on $(A_1, \ldots, A_{n-1})$ // Slice 6
for $i = 1$ to $n-1$ **do** // Slice 7
  CNOT(A_i, B_i)
end

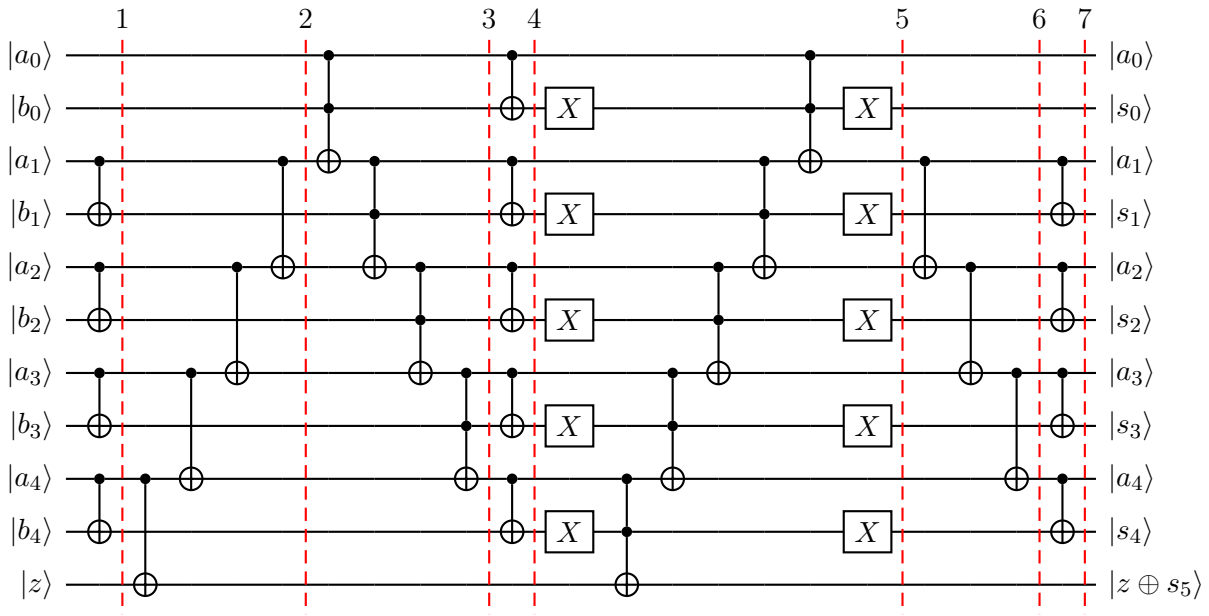


Figure 4.3: Quantum ripple-carry adder where $n = 5$.

Theorem 4. *Let U be the operator performing the in-place addition of two n-bit integers a and b:*

$$U\left|a\right\rangle\left|b\right\rangle\left|z\right\rangle = \left|a\right\rangle\left|a+b \mod 2^n\right\rangle\left|z \oplus (a+b)_n\right\rangle. \tag{4.35}$$

U can be implemented over the $\{Toffoli, CNOT, X\}$ gate set with a depth of $\mathcal{O}(\log_2^2(n))$ and a gate count of $\mathcal{O}(n\log_2(n))$, without any ancillary qubits.

Proof. We prove Theorem 4 by demonstrating that each slice of the circuit produced by Algorithm 3, which implements the in-place addition between a and b, can be implemented over the $\{\text{Toffoli}, \text{CNOT}, X\}$ gate set with depth and gate count complexities of at most $\mathcal{O}(\log_2^2(n))$ and $\mathcal{O}(n\log_2(n))$, respectively. Slices 1, 4, and 7 are each composed of a single layer of parallel CNOT gates, inducing a depth of $\mathcal{O}(1)$ and a gate count of $\mathcal{O}(n)$. Slices 2 and 6 can be implemented with a depth of $\mathcal{O}(\log_2(n))$ and a gate count of $\mathcal{O}(n)$ using only CNOT gates, as proven in Theorem 1. Finally, Slices 3 and 5 can be implemented with a depth of $\mathcal{O}(\log_2^2(n))$ and a gate count of $\mathcal{O}(n\log_2(n))$ over the $\{\text{Toffoli}, X\}$ gate set, as proven in Theorem 3. □

4.4 Extension to controlled addition

Algorithm 4: Ancilla-free controlled ripple-carry adder

Input: A control qubit $\left|c\right\rangle_C$, two n-qubit registers $\left|a\right\rangle_A$ and $\left|b\right\rangle_B$, and a single-qubit register $\left|z\right\rangle_Z$.

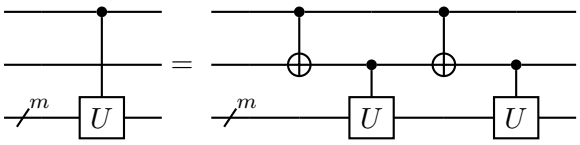
Output: A circuit implementing the in-place controlled addition of a and b: $\left|a\right\rangle_A\left|ca+b \mod 2^n\right\rangle_B\left|z \oplus c(a+b)_n\right\rangle_Z$.

1 **for** $i = 1$ to $n-1$ **do** // Slice 1
2 | CNOT(A_i, B_i)
3 **end**
4 Toffoli(C, A_{n-1}, Z) // Slice 2
5 Apply $\mathcal{L}_1^{(n-2)}$ on $(A_1, \ldots, A_{n-1})$
6 Apply $\left(\mathcal{L}_2^{(n-1)}\right)^\dagger$ on $(A_0, B_0, \ldots, A_{n-2}, B_{n-2}, A_{n-1})$ // Slice 3
7 Apply $\mathcal{F}_2^{(n)}$ on $(C, A_0, B_0, \ldots, A_{n-1}, B_{n-1})$ // Slice 4
8 Apply $\mathcal{F}_1^{(n)}$ on $(C, B_0, \ldots, B_{n-1})$ // Slice 5
9 MCX$_3(C, A_{n-1}, B_{n-1}, Z)$
10 Apply $\mathcal{L}_2^{(n-1)}$ on $(A_0, B_0, \ldots, A_{n-2}, B_{n-2}, A_{n-1})$
11 Apply $\mathcal{F}_1^{(n)}$ on $(C, B_0, \ldots, B_{n-1})$
12 Apply $\left(\mathcal{L}_1^{(n-2)}\right)^\dagger$ on $(A_1, \ldots, A_{n-1})$ // Slice 6
13 **for** $i = 1$ to $n-1$ **do** // Slice 7
14 | CNOT(A_i, B_i)
15 **end**

Based on the results established in Section 4.3, we present a controlled adder with the same asymptotic depth and gate count complexities as Theorem 4, and which also does not use any ancillary qubits. We will rely on the following lemma, which states that the $\mathcal{F}_2^{(n)}$ operator, introduced in Section 1.2.3.2, can be implemented with the same asymptotic depth and gate count complexities as the $\mathcal{F}_1^{(n)}$ operator.

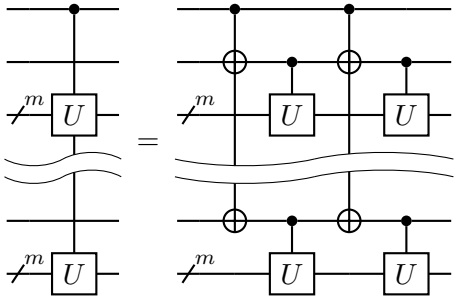
Lemma 1. *The $\mathcal{F}_2^{(n)}$ operator can be implemented over the $\{Toffoli, CNOT\}$ gate set with a depth of $\mathcal{O}(\log_2(n))$ and a gate count of $\mathcal{O}(n)$, without any ancillary qubits.*

Proof. We rely on the following equality, which is for example used in References [25, 26]:



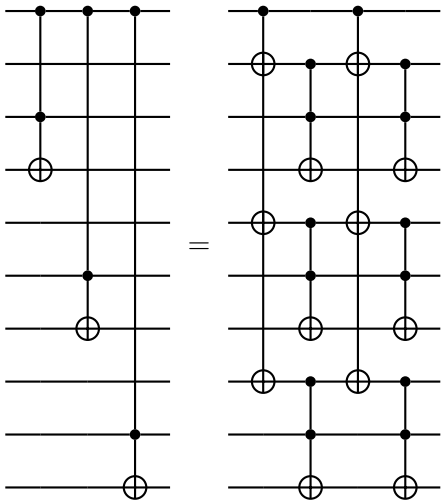
(4.36)

where $U^2 = I$. Based on this equality, we can derive the following equality:



(4.37)

where $U^2 = I$. Notice that in the case where $m = 2$ and $U = \text{CNOT}$, this circuit implements the $\mathcal{F}_2^{(n)}$ operator using n dirty ancillary qubits. For example, in the case where $n = 3$, we get the following circuit equality for the implementation of the $\mathcal{F}_2^{(3)}$ operator using 3 dirty ancillary qubits:



(4.38)

As such, the $\mathcal{F}_2^{(n)}$ operator can be implemented by two layers of parallel Toffoli gates and two $\mathcal{F}_1^{(n)}$ operators, using n dirty ancillary qubits.

In the case where $n = 1$, the $\mathcal{F}_2^{(n)}$ operator can be implemented with a single Toffoli gate. In the more general case where $n > 1$, the $\mathcal{F}_2^{(n)}$ operator can be split into two operators which can be implemented sequentially: $\mathcal{F}_2^{(\lceil n/2 \rceil)}$ and $\mathcal{F}_2^{(\lfloor n/2 \rfloor)}$. The operators $\mathcal{F}_2^{(\lceil n/2 \rceil)}$ and $\mathcal{F}_2^{(\lfloor n/2 \rfloor)}$ are both not acting on at least $\lceil n/2 \rceil$ qubits, which can be used as dirty ancillary qubits to implement the operators as in the right-hand side of Equation 4.37. This results in a circuit implementing

the $\mathcal{F}_2^{(n)}$ operator with 4 layers of parallel Toffoli gates, two $\mathcal{F}_1^{(\lceil n/2 \rceil)}$ operators and two $\mathcal{F}_1^{(\lfloor n/2 \rfloor)}$ operators. The $\mathcal{F}_1^{(\lceil n/2 \rceil)}$ and $\mathcal{F}_1^{(\lfloor n/2 \rfloor)}$ operators can be implemented with a depth of $\mathcal{O}(\log_2(n))$ using $\mathcal{O}(n)$ CNOT gates [27, 28]. Thus, the $\mathcal{F}_2^{(n)}$ operator can be implemented with a depth of $\mathcal{O}(\log_2(n))$ and a gate count of $\mathcal{O}(n)$, without any ancillary qubits. □

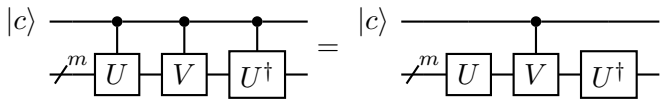
Algorithm 4 presents a pseudo-code for generating a circuit implementing the controlled addition operator. The structure of this algorithm is similar to that of Algorithm 3, as indicated by the slices in the pseudo-code. The correctness and the depth and gate count complexities of the circuit produced by Algorithm 4 are established by the following theorem, which generalizes the result of Theorem 4 to a controlled adder.

Theorem 5. *Let U be the operator performing the in-place controlled addition of two n-bit integers a and b:*

$$U\left|c\right\rangle\left|a\right\rangle\left|b\right\rangle\left|z\right\rangle = \left|c\right\rangle\left|a\right\rangle\left|ca+b \mod 2^n\right\rangle\left|z \oplus c(a+b)_n\right\rangle. \tag{4.39}$$

U can be implemented over the $\{Toffoli, CNOT, X\}$ gate set with a depth of $\mathcal{O}(\log_2^2(n))$ and a gate count of $\mathcal{O}(n\log_2(n))$, without any ancillary qubits.

Proof. We will rely on the following equality:



(4.40)

which can be easily proven by case distinction:

- In the case where $c = 1$, the unitaries U, V, and $U^\dagger$ are applied in both circuits.
- In the case where $c = 0$, the left-hand side circuit is equal to the identity since no unitaries are applied. On the right-hand side, only U and $U^\dagger$ are applied, which cancel each other out because $UU^\dagger = I$.

An adder controlled by a qubit C can be constructed from the circuit produced by Algorithm 3 simply by adding C as a control to all the gates in the circuit. The CNOT circuit constructed in the Slice 7 of Algorithm 3 is the dagger of the CNOT circuit constructed in the Slice 1 of Algorithm 3. Therefore, as indicated by Equation 4.40, these CNOT gates do not need to be controlled by C, as done in Algorithm 4. In the Slice 2 of Algorithm 3, the $\mathcal{L}_1^{(n-1)}$ operator applied on the qubits $(A_1, \ldots, A_{n-1}, Z)$ can be implemented by first applying a CNOT gate on the qubits (A_{n-1}, Z), and then applying the $\mathcal{L}_1^{(n-2)}$ on the qubits $(A_1, \ldots, A_{n-1})$. Adding the qubit C as a control of the CNOT gate results in a Toffoli gate applied on the qubits (C, A_{n-1}, Z), as done in Algorithm 4. The $\mathcal{L}_1^{(n-2)}$ operator applied on the qubits $(A_1, \ldots, A_{n-1})$ is the dagger of the $\left(\mathcal{L}_1^{(n-2)}\right)^\dagger$ operator applied in the Slice 6. Therefore, as indicated by Equation 4.40, these two operators do not need to be controlled by C, as done in Algorithm 4. Similarly, in the Slice 5 of Algorithm 3, the $\mathcal{L}_2^{(n)}$ operator applied on the qubits $(A_0, B_0, \ldots, A_{n-1}, B_{n-1}, Z)$ can be implemented by first applying a Toffoli gate on the qubits (A_{n-1}, B_{n-1}, Z), and then applying the $\mathcal{L}_2^{(n-1)}$ operator on the qubits $(A_0, B_0, \ldots, A_{n-2}, B_{n-2}, A_{n-1})$. Adding the qubit C as a control of the Toffoli gate results in a MCX_3 gate applied on the qubits (C, A_{n-1}, B_{n-1}, Z), as done in Algorithm 4. The $\mathcal{L}_2^{(n-1)}$ operator applied on the qubits $(A_0, B_0, \ldots, A_{n-2}, B_{n-2}, A_{n-1})$ is the

dagger of the $\left(\mathcal{L}_2^{(n-1)}\right)^\dagger$ operator applied in the Slice 3. Therefore, as indicated by Equation 4.40, these two operators do not need to be controlled by C, as done in Algorithm 4. However, the X gates applied on the qubits B_i in the Slice 5 of Algorithm 3 must be controlled by C, which corresponds to the $\mathcal{F}_1^{(n-1)}$ operator applied on $(C, B_1, \ldots, B_{n-1})$ in Algorithm 4. Finally, adding C as a controlled of the CNOT gates applied in the Slice 4 of Algorithm 3 corresponds to applying the operator $\mathcal{F}_2^{(n)}$ on the qubits $(C, A_0, B_0, \ldots, A_{n-1}, B_{n-1})$, as done in Algorithm 4. Thus, the circuit produced by Algorithm 4 is equivalent to the circuit produced by Algorithm 3 controlled by a qubit C, and therefore implements the controlled addition operator.

The $\mathcal{L}_1^{(n-2)}$ and $\left(\mathcal{L}_1^{(n-2)}\right)^\dagger$ operators in Slices 2 and 6 can be implemented with a depth of $\mathcal{O}(\log_2(n))$ and a CNOT-count of $\mathcal{O}(n)$, as stated by Theorem 1. The $\mathcal{L}_2^{(n-1)}$ and $\left(\mathcal{L}_2^{(n-1)}\right)^\dagger$ operators in Slices 3 and 5 can be implemented with a depth of $\mathcal{O}(\log_2^2(n))$ and a CNOT-count of $\mathcal{O}(n\log_2^2(n))$, as stated by Theorem 3. The two $\mathcal{F}_1^{(n)}$ operators in Slice 5 can be implemented with a depth of $\mathcal{O}(\log_2(n))$ and a CNOT-count of $\mathcal{O}(n)$ [27,28]. The $\mathcal{F}_2^{(n)}$ operator in Slice 4 can be implemented with a depth of $\mathcal{O}(\log_2(n))$ and a CNOT-count of $\mathcal{O}(n)$, as stated by Lemma 1. The Toffoli gate in Slice 2 and the MCX_3 gate in Slice 5 can be implemented in constant depth and with a constant number of gates. The two subcircuits formed by the remaining CNOT gates in Slices 1 and 7 both have a constant depth equal to 1, as all the CNOT gates are applied on different qubits. Thus, the circuit produced by Algorithm 4 can be implemented over the $\{\mathrm{Toffoli}, \mathrm{CNOT}, X\}$ gate set with a depth of $\mathcal{O}(\log_2^2(n))$ and a gate count is $\mathcal{O}(n\log_2(n))$, and without any ancillary qubits. □

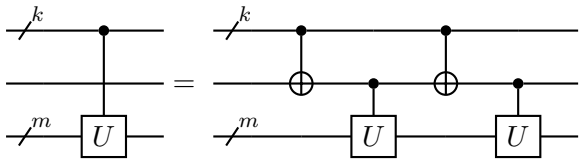
An algorithm for constructing a circuit that implements an adder controlled by multiple qubits can easily be derived from Algorithm 4. The resulting depth complexity overhead scales logarithmically with the number of controlled qubits, as stated by the following corollary.

Corollary 2. *Let U be the operator performing the in-place addition of two n-bit integers a and b controlled by $c \in \{0,1\}^k$:*

$$|c\rangle\,|a\rangle\,|b\rangle\,|z\rangle \mapsto |c\rangle\,|a\rangle\,|\left(\prod_{i=0}^{k-1} c_i x\right) a + b \mod 2^n\rangle|z \oplus \left(\prod_{i=0}^{k-1} c_i\right)(a+b)_n\rangle \tag{4.41}$$

U can be implemented over the $\{$Toffoli, CNOT, $X\}$ gate set with a depth of $\mathcal{O}(\log_2^2(n)+\log_2(k))$ and a gate count of $\mathcal{O}(n\log_2(n)+k)$, without any ancillary qubits.

Proof. Let K be the set of k qubits by which the adder is controlled. Replacing each operator U controlled by the qubit C in Algorithm 4 with U controlled by the qubits K results in a circuit implementing an adder controlled by the qubits K. An operator U controlled by the qubits K can be implemented as follows, by using two MCX_k gates, two U gates controlled by a single-qubit, and a dirty ancillary qubit:



(4.42)

This equality is similar to Equation 4.36, but with k control qubits. A dirty ancillary qubit is always available since all the operators acting on the control qubit C in Algorithm 4 do not

act on all the qubits. Moreover, the two MCX_k gates can be implemented over the $\{\text{Toffoli}, X\}$ gate set with a depth of $\mathcal{O}(\log_2(k))$ and a gate count of $\mathcal{O}(k)$ [26], since at least two dirty qubits are available (because the MCX_k gates do not act on the qubits on which U is applied). Furthermore, as stated by Theorem 5, each gate U controlled by C in Algorithm 4 can be implemented over the $\{\text{Toffoli}, \text{CNOT}, X\}$ gate set with a depth of $\mathcal{O}(\log_2^2(n))$ and a gate count of $\mathcal{O}(n \log_2(n))$. And there is only a constant number of operators in Algorithm 4 for which C acts as a control qubit. Thus, the resulting circuit implements an adder controlled by the qubits K over the $\{\text{Toffoli}, \text{CNOT}, X\}$ gate set, with a depth and gate count of $\mathcal{O}(\log_2^2(n) + \log_2(n))$ and $\mathcal{O}(n \log_2(n) + k)$ respectively, and without any ancillary qubits. □

Chapter 5

Quantum Binary Field Multiplication with Low Space-Time Cost

Binary field arithmetic plays a crucial role in the design of quantum polynomial-time attacks on elliptic curve cryptography over binary fields [15, 86, 87]. Consequently, much work has been devoted to analyzing and optimizing the arithmetic operations required for solving the discrete logarithm problem on binary elliptic curves [88–94]. Among these operations, multiplication of elements in binary fields stands out as one of the most computationally expensive. As such, various techniques have been developed to construct efficient quantum circuits performing binary field multiplication. In Reference [87], a linear depth circuit composed of $\mathcal{O}(n^2)$ Toffoli gates is presented, based on the classical Mastrovito multiplier [95, 96]. It was demonstrated in Reference [97] that the depth of the circuit could be reduced to $\mathcal{O}(\log_2(n))$, at the cost of $\mathcal{O}(n^2)$ ancillary qubits. Subsequent research has focused on reducing the number of Toffoli gates, as they are particularly costly to implement fault-tolerantly. Notable progress has been made using approaches inspired by the classical Karatsuba multiplication algorithm [98], which led to a Toffoli gate count of $\mathcal{O}(n^{\log_2(3)})$ [99–101]. However, this subquadratic Toffoli gate count comes with trade-offs: either in space, by requiring significantly more qubits, or in time, by substantially increasing the circuit depth. As a result, despite achieving a subquadratic number of Toffoli gates, these circuits introduce a significant overhead in space-time cost. Notably, the space-time costs of these circuits, despite having a subquadratic number of Toffoli gates, exceed those of the circuits containing a quadratic number of Toffoli gates proposed in References [87, 97]

In this chapter, we propose a novel approach, also similar to the Karatsuba multiplication algorithm, to achieve a Toffoli gate count of $\mathcal{O}(n^{\log_2(3)})$. Our method offers improved parallelization of the circuit when using ancillary qubits. In particular, we demonstrate that a linear depth can be achieved with $\mathcal{O}(n \log_2(n))$ ancillary qubits. This results in a quantum circuit for binary field multiplication that provides the best known space-time trade-off while maintaining the subquadratic number of Toffoli gates. Moreover, we show that for certain families of primitive polynomials, namely trinomials and equally spaced primitive polynomials, logarithmic depth can be achieved using $\mathcal{O}(n^{\log_2(3)})$ ancillary qubits. This leads to an even better space-time cost for these specific cases. Table 5.1 summarizes the cost of different methods over various metrics.

Chapter organization. In Section 5.1, we present the key concepts and notations for binary field multiplication. Then, in Section 5.2, we detail our algorithm for binary field multiplication with linear depth and subquadratic Toffoli gate count. Finally, in Section 5.3, we demonstrate how to achieve logarithmic depth for trinomials and equally spaced primitive polynomials. Our

Reference	Toffoli-count	Qubit-count	Depth	Space-time cost
Arbitrary primitive polynomials				
Ref. [87]	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$
Ref. [97]	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(\log_2(n))$	$\mathcal{O}(n^2 \log_2(n))$
Ref. [99]	$\mathcal{O}(n^{\log_2(3)})$	$\mathcal{O}(n^{\log_2(3)})$	$\mathcal{O}(n)$	$\mathcal{O}(n^{2.58})$
Ref. [100, 101]	$\mathcal{O}(n^{\log_2(3)})$	$\mathcal{O}(n)$	$\mathcal{O}(n^{\log_2(3)})$	$\mathcal{O}(n^{2.58})$
Section 5.2	$\mathcal{O}(n^{\log_2(3)})$	$\mathcal{O}(n \log_2(n))$	$\mathcal{O}(n)$	$\mathcal{O}(n^2 \log_2(n))$
Trinomials and equally spaced primitive polynomials				
Section 5.3	$\mathcal{O}(n^{\log_2(3)})$	$\mathcal{O}(n^{\log_2(3)})$	$\mathcal{O}(\log_2(n))$	$\mathcal{O}(n^{1.58} \log_2(n))$

Table 5.1: Toffoli-count, qubit-count, depth and space-time cost comparison between different quantum circuits performing multiplication over $GF(2^n)$.

implementation of the proposed algorithm is publicly available [9].

5.1 Binary field multiplication

An element of $GF(2^n)$ can be represented by a polynomial $A(x)$ with binary coefficients and degree less than n:

$$A(x) = \sum_{i=0}^{n-1} a_i x^i \tag{5.1}$$

where $a_i \in GF(2)$ for all i. Such polynomial $A(x)$ can be encoded into a binary vector $\boldsymbol{a}$ of size n representing its coefficients:

$$\boldsymbol{a} = \begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \end{bmatrix}. \tag{5.2}$$

The multiplication of two polynomials $A(x)$ and $B(x)$ in $GF(2^n)$ is done modulo an irreducible primitive polynomial $P(x)$ of degree n:

$$P(x) = \sum_{i=0}^{n-1} p_i x^i + x^n \tag{5.3}$$

where $p_i \in GF(2)$ for all i. As proven in Reference [102], multiplication over the field $GF(2^n)$ corresponds to the following operation:

$$|\boldsymbol{a}\rangle |\boldsymbol{b}\rangle |\boldsymbol{0}\rangle \mapsto |\boldsymbol{a}\rangle |\boldsymbol{b}\rangle |\boldsymbol{d} \oplus Q\boldsymbol{e}\rangle \tag{5.4}$$

where all three registers are of size n, Q is a binary matrix of size $n \times n-1$, and where $\boldsymbol{d}$ and $\boldsymbol{e}$ are vectors defined as

$$\begin{aligned} \boldsymbol{d} &= L\boldsymbol{b} \\ \boldsymbol{e} &= U\boldsymbol{b}, \end{aligned} \tag{5.5}$$

with L and U defined as

$$L = \begin{bmatrix} a_0 & 0 & \dots & 0 & 0 \\ a_1 & a_0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-2} & a_{n-3} & \dots & a_0 & 0 \\ a_{n-1} & a_{n-2} & \dots & a_1 & a_0 \end{bmatrix}, \quad U = \begin{bmatrix} 0 & a_{n-1} & a_{n-2} & \dots & a_2 & a_1 \\ 0 & 0 & a_{n-1} & \dots & a_3 & a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & a_{n-1} & a_{n-2} \\ 0 & 0 & 0 & \dots & 0 & a_{n-1} \end{bmatrix}. \tag{5.6}$$

The matrix Q, referred to as the reduction matrix, is derived from the primitive polynomial $P(x)$. The key property of Q is that it relates the higher-degree terms in the product to the lower-degree terms through the reduction modulo $P(x)$. This reduction matrix satisfies the following equation:

$$\boldsymbol{x}^{\uparrow} \equiv Q^T \boldsymbol{x}^{\downarrow} \pmod{P(x)} \tag{5.7}$$

where $\boldsymbol{x}^{\downarrow} = [1, x, x^2, \dots, x^{n-1}]^T$ and $\boldsymbol{x}^{\uparrow} = [x^n, x^{n+1}, \dots, x^{2n-2}]^T$. Using the fact that $P(x) \equiv 0 \pmod{P(x)}$, we can deduce from Equation (5.3) that

$$x^n \equiv \sum_{i=0}^{n-1} p_i x^i \pmod{P(x)}. \tag{5.8}$$

Hence, the first column of the reduction matrix Q represents the coefficients p_i of the primitive polynomial $P(x)$. The subsequent columns of Q can be easily computed by induction as $x^{n+i} \equiv x \cdot x^{n+i-1} \pmod{P(x)}$, for all integers $i > 0$.

Example. As an example, for the field $G(2^7)$ with primitive polynomial $P(x) = x^7 + x^5 + x^3 + x + 1$, the reduction matrix Q is

$$Q = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{5.9}$$

The multiplication of the polynomials $A(x) = x^5 + x^3 + 1$ and $B(x) = x^2 + x$ with this primitive polynomial $P(x)$ is

$$\begin{aligned} (x^5 + x^3 + 1)(x^2 + x) \pmod{P(x)} &= x^7 + x^5 + x^2 + x^6 + x^4 + x \pmod{P(x)} \\ &= x^6 + x^4 + x^3 + x^2 + 1 \pmod{P(x)} \end{aligned} \tag{5.10}$$

The binary vectors $\boldsymbol{a}, \boldsymbol{b}$ representing the polynomials $A(x)$ and $B(x)$ respectively, and the matrix L and U are:

$$\boldsymbol{a} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{5.11}$$

This leads to the following values for the vectors $\boldsymbol{e}$ and $\boldsymbol{d}$:

$$\boldsymbol{d} = L\boldsymbol{b} = \begin{bmatrix}0 & 1 & 1 & 0 & 1 & 1 & 1\end{bmatrix}^T, \quad \boldsymbol{e} = U\boldsymbol{b} = \begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0\end{bmatrix}^T. \tag{5.12}$$

Finally, we have:

$$\boldsymbol{d} \oplus Q\boldsymbol{e} = \begin{bmatrix}1 & 0 & 1 & 1 & 1 & 0 & 1\end{bmatrix}^T \tag{5.13}$$

which represents the polynomial $x^6 + x^4 + x^3 + x^2 + 1$, matching the result obtained earlier in Equation (5.10).

5.2 Synthesis algorithm for binary field multiplication

In this section, we present methods for a synthesizing binary field multiplier circuit using a subquadratic number of Toffoli gates. First, in Subsection 5.2.1, we introduce an algorithm for constructing a circuit that performs multiplication over $GF(2^n)$ using $\mathcal{O}(n^{\log_2(3)})$ Toffoli gates, without requiring any ancillary qubits. Then, in Subsection 5.2.2, we demonstrate how our approach can be adapted to construct a circuit for multiplication over $GF(2^n)$ with linear depth by using $\mathcal{O}(n \log_2(n))$ ancillary qubits, while maintaining the same subquadratic number of Toffoli gates.

5.2.1 Subquadratic Toffoli gate count

Reference [87] outlines a straightforward approach for implementing the following operation:

$$|\boldsymbol{a}\rangle |\boldsymbol{b}\rangle |\boldsymbol{0}\rangle \mapsto |\boldsymbol{a}\rangle |\boldsymbol{b}\rangle |\boldsymbol{d} \oplus Q\boldsymbol{e}\rangle \tag{5.14}$$

which corresponds to multiplication over $GF(2^n)$, as described in Section 5.1. This method consists of three stages:

1. First, the transformation $|\boldsymbol{a}\rangle |\boldsymbol{b}\rangle |\boldsymbol{0}\rangle \mapsto |\boldsymbol{a}\rangle |\boldsymbol{b}\rangle |\boldsymbol{e}\rangle$ is achieved using Toffoli gates, where each gate has controls on the registers $\boldsymbol{a}$ and $\boldsymbol{b}$, and targets the third register.

2. Then, the transformation $|\boldsymbol{a}\rangle |\boldsymbol{b}\rangle |\boldsymbol{e}\rangle \mapsto |\boldsymbol{a}\rangle |\boldsymbol{b}\rangle |Q\boldsymbol{e}\rangle$ is realized via a CNOT circuit on the third register.

3. Similarly to the first step, the final transformation $|\boldsymbol{a}\rangle |\boldsymbol{b}\rangle |\boldsymbol{d} \oplus Q\boldsymbol{e}\rangle \mapsto |\boldsymbol{a}\rangle |\boldsymbol{b}\rangle |\boldsymbol{d} \oplus Q\boldsymbol{e}\rangle$ is achieved using Toffoli gates, where each gate is has controls on the registers $\boldsymbol{a}$ and $\boldsymbol{b}$, and targets the third register.

This method yields a circuit containing $\mathcal{O}(n^2)$ Toffoli gates for the first and last steps, and $\mathcal{O}(n^2)$ CNOT gates for the middle CNOT circuit. This results in a circuit having a better Toffoli gate count than the naive approach using $\mathcal{O}(n^3)$ Toffoli gates.

To achieve a subquadratic Toffoli gate count, we will formalize the problem as a CCZ gate count optimization problem over the $\{\text{CNOT}, CCZ\}$ gate set. We can notice that the mapping of Equation 5.14 can be realized using only Toffoli gates, with targets on the third register and controls on the first two registers $\boldsymbol{a}$ and $\boldsymbol{b}$. Then, by using the circuit equality of Equation 1.60, we can conjugate all the Toffoli gates by a Hadamard gate on their target qubit. Adjacent pairs of Hadamard can then be removed from the circuit by using the fact that the Hadamard gate is self-inverse. This results in a circuit composed of a layer of Hadamard gates on the third register, followed by a layer of CCZ gates associated with $\boldsymbol{d}$ and $Q\boldsymbol{e}$, and then another layer of

Hadamard gates on the third register. The resulting CCZ subcircuit can then be represented by a third-order multilinear homogeneous polynomial with binary coefficients, denoted f, satisfying the following equation:

$$f(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{c}') = g(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) \oplus h(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}') \tag{5.15}$$

where $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ are the three registers and $\boldsymbol{c}' = Q^{-1}\boldsymbol{c}$, and where g and h are third-order multilinear homogeneous polynomials respectively associated with $\boldsymbol{d}$ and $Q\boldsymbol{e}$ and defined as follows:

$$g(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) = \sum_{i=0}^{n-1} \sum_{j=0}^{i} a_j b_{i-j} c_i \pmod 2, \tag{5.16}$$

$$h(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}') = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} a_j b_{n+i-j} c'_i \pmod 2. \tag{5.17}$$

Finding an implementation of the $GF(2^n)$ multiplier with a low number of CCZ (or Toffoli) gates then consists in performing the synthesis of the polynomial f with an optimized number of CCZ gates. To do so, we will rely on the following theorem, which defines a recursive formula for computing Equation 5.15. This recursive formula have some similarities with the Karatsuba method for performing fast regular multiplications [98].

Theorem 6. *Let $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ and $\boldsymbol{c}'$ be vectors of size n, where n is an even integer. Then,*

$$\begin{aligned} g(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) \oplus h(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}') = {} & g(\boldsymbol{a}_L \oplus \boldsymbol{a}_R, \boldsymbol{b}_L \oplus \boldsymbol{b}_R, \boldsymbol{c}_R) \oplus h(\boldsymbol{a}_L \oplus \boldsymbol{a}_R, \boldsymbol{b}_L \oplus \boldsymbol{b}_R, \boldsymbol{c}'_L) \\ & \oplus g(\boldsymbol{a}_R, \boldsymbol{b}_R, \boldsymbol{c}'_L \oplus \boldsymbol{c}_R) \oplus h(\boldsymbol{a}_R, \boldsymbol{b}_R, \boldsymbol{c}'_L \oplus \boldsymbol{c}'_R) \\ & \oplus g(\boldsymbol{a}_L, \boldsymbol{b}_L, \boldsymbol{c}_L \oplus \boldsymbol{c}_R) \oplus h(\boldsymbol{a}_L, \boldsymbol{b}_L, \boldsymbol{c}'_L \oplus \boldsymbol{c}_R) \end{aligned} \tag{5.18}$$

where the vectors $\boldsymbol{a}_L$ and $\boldsymbol{a}_R$ are respectively equal to the left and right halves of the vector $\boldsymbol{a}$.

Proof. Equation 5.18 holds if and only if the two following equations are true:

$$\begin{aligned} g(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) = {} & g(\boldsymbol{a}_L \oplus \boldsymbol{a}_R, \boldsymbol{b}_L \oplus \boldsymbol{b}_R, \boldsymbol{c}_R) \oplus g(\boldsymbol{a}_R, \boldsymbol{b}_R, \boldsymbol{c}_R) \\ & \oplus g(\boldsymbol{a}_L, \boldsymbol{b}_L, \boldsymbol{c}_L \oplus \boldsymbol{c}_R) \oplus h(\boldsymbol{a}_L, \boldsymbol{b}_L, \boldsymbol{c}_R) \end{aligned} \tag{5.19}$$

$$\begin{aligned} h(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}') = {} & h(\boldsymbol{a}_L \oplus \boldsymbol{a}_R, \boldsymbol{b}_L \oplus \boldsymbol{b}_R, \boldsymbol{c}'_L) \oplus g(\boldsymbol{a}_R, \boldsymbol{b}_R, \boldsymbol{c}'_L) \\ & \oplus h(\boldsymbol{a}_R, \boldsymbol{b}_R, \boldsymbol{c}'_L \oplus \boldsymbol{c}'_R) \oplus h(\boldsymbol{a}_L, \boldsymbol{b}_L, \boldsymbol{c}'_L) \end{aligned} \tag{5.20}$$

We first prove Equation 5.19:

$$
\begin{aligned}
&g(\boldsymbol{a}_L \oplus \boldsymbol{a}_R, \boldsymbol{b}_L \oplus \boldsymbol{b}_R, \boldsymbol{c}_R) \oplus g(\boldsymbol{a}_R, \boldsymbol{b}_R, \boldsymbol{c}_R) \oplus g(\boldsymbol{a}_L, \boldsymbol{b}_L, \boldsymbol{c}_L \oplus \boldsymbol{c}_R) \oplus h(\boldsymbol{a}_L, \boldsymbol{b}_L, \boldsymbol{c}_R) \\
&= \sum_{i=0}^{\frac{n}{2}-1} \sum_{j=0}^{i} (a_j + a_{\frac{n}{2}+j})(b_{i-j} + b_{\frac{n}{2}+i-j}) c_{\frac{n}{2}+i} + \sum_{i=0}^{\frac{n}{2}-1} \sum_{j=0}^{i} a_{\frac{n}{2}+j} b_{\frac{n}{2}+i-j} c_{\frac{n}{2}+i} \\
&\quad + \sum_{i=0}^{\frac{n}{2}-1} \sum_{j=0}^{i} a_j b_{i-j} (c_i + c_{\frac{n}{2}+i}) + \sum_{i=0}^{\frac{n}{2}-2} \sum_{j=i+1}^{\frac{n}{2}-1} a_j b_{\frac{n}{2}+i-j} c_{\frac{n}{2}+i} \pmod 2 \\
&= \sum_{i=0}^{\frac{n}{2}-1} \sum_{j=0}^{i} a_j b_{\frac{n}{2}+i-j} c_{\frac{n}{2}+i} + \sum_{i=0}^{\frac{n}{2}-1} \sum_{j=0}^{i} a_{\frac{n}{2}+j} b_{i-j} c_{\frac{n}{2}+i} + \sum_{i=0}^{\frac{n}{2}-1} \sum_{j=0}^{i} a_j b_{i-j} c_i \\
&\quad + \sum_{i=0}^{\frac{n}{2}-2} \sum_{j=i+1}^{\frac{n}{2}-1} a_j b_{\frac{n}{2}+i-j} c_{\frac{n}{2}+i} \pmod 2 \\
&= \sum_{i=\frac{n}{2}}^{n-1} \sum_{j=0}^{i-\frac{n}{2}} a_j b_{i-j} c_i + \sum_{i=\frac{n}{2}}^{n-1} \sum_{j=\frac{n}{2}}^{i} a_j b_{i-j} c_i + \sum_{i=0}^{\frac{n}{2}-1} \sum_{j=0}^{i} a_j b_{i-j} c_i \qquad (5.21) \\
&\quad + \sum_{i=\frac{n}{2}}^{n-2} \sum_{j=i-\frac{n}{2}+1}^{\frac{n}{2}-1} a_j b_{i-j} c_i \pmod 2 \\
&= \sum_{i=\frac{n}{2}}^{n-1} \sum_{j=0}^{\frac{n}{2}-1} a_j b_{i-j} c_i + \sum_{i=\frac{n}{2}}^{n-1} \sum_{j=\frac{n}{2}}^{i} a_j b_{i-j} c_i + \sum_{i=0}^{\frac{n}{2}-1} \sum_{j=0}^{i} a_j b_{i-j} c_i \pmod 2 \\
&= \sum_{i=\frac{n}{2}}^{n-1} \sum_{j=0}^{i} a_j b_{i-j} c_i + \sum_{i=0}^{\frac{n}{2}-1} \sum_{j=0}^{i} a_j b_{i-j} c_i \pmod 2 \\
&= \sum_{i=0}^{n-1} \sum_{j=0}^{i} a_j b_{i-j} c_i \pmod 2 \\
&= g(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}).
\end{aligned}
$$

We now prove Equation 5.20:

$$
\begin{aligned}
&h(\boldsymbol{a}_L \oplus \boldsymbol{a}_R, \boldsymbol{b}_L \oplus \boldsymbol{b}_R, \boldsymbol{c}'_L) \oplus g(\boldsymbol{a}_R, \boldsymbol{b}_R, \boldsymbol{c}'_L) \oplus h(\boldsymbol{a}_R, \boldsymbol{b}_R, \boldsymbol{c}'_L \oplus \boldsymbol{c}'_R) \oplus h(\boldsymbol{a}_L, \boldsymbol{b}_L, \boldsymbol{c}'_L) \\
&= \sum_{i=0}^{\frac{n}{2}-2} \sum_{j=i+1}^{\frac{n}{2}-1} (a_j + a_{\frac{n}{2}+j})(b_{\frac{n}{2}+i-j} + b_{n+i-j})c'_i + \sum_{i=0}^{\frac{n}{2}-1} \sum_{j=0}^{i} a_{\frac{n}{2}+j} b_{\frac{n}{2}+i-j} c'_i \\
&\quad + \sum_{i=0}^{\frac{n}{2}-2} \sum_{j=i+1}^{\frac{n}{2}-1} a_{\frac{n}{2}+j} b_{n+i-j}(c'_i + c'_{\frac{n}{2}+i}) + \sum_{i=0}^{\frac{n}{2}-2} \sum_{j=i+1}^{\frac{n}{2}-1} a_j b_{\frac{n}{2}+i-j} c'_i \pmod 2 \\
&= \sum_{i=0}^{\frac{n}{2}-2} \sum_{j=i+1}^{\frac{n}{2}-1} a_j b_{n+i-j} c'_i + \sum_{i=0}^{\frac{n}{2}-2} \sum_{j=i+1}^{\frac{n}{2}-1} a_{\frac{n}{2}+j} b_{\frac{n}{2}+i-j} c'_i + \sum_{i=0}^{\frac{n}{2}-1} \sum_{j=0}^{i} a_{\frac{n}{2}+j} b_{\frac{n}{2}+i-j} c'_i \\
&\quad + \sum_{i=0}^{\frac{n}{2}-2} \sum_{j=i+1}^{\frac{n}{2}-1} a_{\frac{n}{2}+j} b_{n+i-j} c'_{\frac{n}{2}+i} \pmod 2 \\
&= \sum_{i=0}^{\frac{n}{2}-2} \sum_{j=i+1}^{\frac{n}{2}-1} a_j b_{n+i-j} c'_i + \sum_{i=0}^{\frac{n}{2}-2} \sum_{j=\frac{n}{2}+i+1}^{n-1} a_j b_{n+i-j} c'_i + \sum_{i=0}^{\frac{n}{2}-1} \sum_{j=\frac{n}{2}}^{\frac{n}{2}+i} a_j b_{n+i-j} c'_i \\
&\quad + \sum_{i=\frac{n}{2}}^{n-2} \sum_{j=i+1}^{n-1} a_j b_{n+i-j} c'_i \pmod 2 \\
&= \sum_{i=0}^{\frac{n}{2}-2} \sum_{j=i+1}^{\frac{n}{2}-1} a_j b_{n+i-j} c'_i + \sum_{i=0}^{\frac{n}{2}-2} \sum_{j=\frac{n}{2}+i+1}^{n-1} a_j b_{n+i-j} c'_i + \sum_{i=0}^{\frac{n}{2}-2} \sum_{j=\frac{n}{2}}^{\frac{n}{2}+i} a_j b_{n+i-j} c'_i \\
&\quad + \sum_{i=\frac{n}{2}-1}^{\frac{n}{2}-1} \sum_{j=i+1}^{n-1} a_j b_{n+i-j} c'_i + \sum_{i=\frac{n}{2}}^{n-2} \sum_{j=i+1}^{n-1} a_j b_{n+i-j} c'_i \pmod 2 \\
&= \sum_{i=0}^{\frac{n}{2}-2} \sum_{j=i+1}^{\frac{n}{2}-1} a_j b_{n+i-j} c'_i + \sum_{i=0}^{\frac{n}{2}-2} \sum_{j=\frac{n}{2}+i+1}^{n-1} a_j b_{n+i-j} c'_i + \sum_{i=0}^{\frac{n}{2}-2} \sum_{j=\frac{n}{2}}^{\frac{n}{2}+i} a_j b_{n+i-j} c'_i \\
&\quad + \sum_{i=\frac{n}{2}-1}^{n-2} \sum_{j=i+1}^{n-1} a_j b_{n+i-j} c'_i \pmod 2 \\
&= \sum_{i=0}^{\frac{n}{2}-2} \sum_{j=i+1}^{\frac{n}{2}+i} a_j b_{n+i-j} c'_i + \sum_{i=0}^{\frac{n}{2}-2} \sum_{j=\frac{n}{2}+i+1}^{n-1} a_j b_{n+i-j} c'_i + \sum_{i=\frac{n}{2}-1}^{n-2} \sum_{j=i+1}^{n-1} a_j b_{n+i-j} c'_i \pmod 2 \\
&= \sum_{i=0}^{\frac{n}{2}-2} \sum_{j=i+1}^{n-1} a_j b_{n+i-j} c'_i + \sum_{i=\frac{n}{2}-1}^{n-2} \sum_{j=i+1}^{n-1} a_j b_{n+i-j} c'_i \pmod 2 \\
&= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} a_j b_{n+i-j} c'_i \pmod 2 \\
&= h(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}').
\end{aligned}
\tag{5.22}
$$

□

For the base case where $n = 1$, we can easily deduce from Equations 5.16 and 5.17 that

$$g(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) \oplus h(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}') = a_0 b_0 c_0 \tag{5.23}$$

where $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$ and $\boldsymbol{c}'$ are vectors of size 1, which corresponds to a single CCZ gate on the qubits a_0, b_0 and c_0. To use the formula given in Theorem 6, the size of the vectors, noted n, must be even. The following theorem demonstrates how we can still rely on the recursive formula of Equation 5.18 when n is odd by increasing the size of the vectors by 1.

Theorem 7. *Let $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ and $\boldsymbol{c}'$ be vectors of size n. And let $\tilde{\boldsymbol{a}}, \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{c}}$ and $\tilde{\boldsymbol{c}}'$ be vectors of size $n+1$, defined as follows:*

$$\tilde{\boldsymbol{a}} = \begin{bmatrix} \boldsymbol{a} \\ 0 \end{bmatrix}, \quad \tilde{\boldsymbol{b}} = \begin{bmatrix} \boldsymbol{b} \\ 0 \end{bmatrix}, \quad \tilde{\boldsymbol{c}} = \begin{bmatrix} \boldsymbol{c} \\ c'_0 \end{bmatrix}, \quad \tilde{\boldsymbol{c}}' = \begin{bmatrix} c'_1 \\ \vdots \\ c'_{n-1} \\ 0 \\ 0 \end{bmatrix}. \tag{5.24}$$

Then,

$$g(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) \oplus h(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}') = g(\tilde{\boldsymbol{a}}, \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{c}}) \oplus h(\tilde{\boldsymbol{a}}, \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{c}}'). \tag{5.25}$$

Proof.

$$\begin{aligned}
&g(\tilde{\boldsymbol{a}}, \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{c}}) \oplus h(\tilde{\boldsymbol{a}}, \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{c}}') \\
&= \sum_{i=0}^{n} \sum_{j=0}^{i} \tilde{a}_j \tilde{b}_{i-j} \tilde{c}_i + \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} \tilde{a}_j \tilde{b}_{n+1+i-j} \tilde{c}'_i \pmod 2 \\
&= \sum_{i=0}^{n-1} \sum_{j=0}^{i} a_j b_{i-j} c_i + \sum_{j=1}^{n-1} a_j b_{n-j} c'_0 + \sum_{i=0}^{n-3} \sum_{j=i+2}^{n-1} a_j b_{n+1+i-j} c'_{i+1} \pmod 2 \\
&= g(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) + \sum_{j=1}^{n-1} a_j b_{n-j} c'_0 + \sum_{i=1}^{n-2} \sum_{j=i+1}^{n-1} a_j b_{n+i-j} c'_i \pmod 2 \\
&= g(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) + \sum_{i=0}^{0} \sum_{j=i+1}^{n-1} a_j b_{n+i-j} c'_i + \sum_{i=1}^{n-2} \sum_{j=i+1}^{n-1} a_j b_{n+i-j} c'_i \pmod 2 \\
&= g(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) + \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} a_j b_{n+i-j} c'_i \pmod 2 \\
&= g(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) \oplus h(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}')
\end{aligned} \tag{5.26}$$

□

Complexity analysis. Theorem 6 divides the sum of g and h with vectors of size n into 3 smaller sums, each involving vectors of size $n/2$. Thus, by applying the equality of Theorem 6 recursively, the sum of g and h with vectors of size n will be divided into $3^{\lceil \log_2(n) \rceil}$ sums of g and h with vectors of single elements, such as in Equation 5.23. Therefore, the constructed circuit performs multiplication over the field $GF(2^n)$ with no more than $3^{\lceil \log_2(n) \rceil}$ CCZ gates. This correponds to $\mathcal{O}(n^{\log_2(3)}) \approx \mathcal{O}(n^{1.58})$ CCZ gates, without requiring any ancillary qubits.

5.2.2 Linear depth

In this section, we demonstrate how Theorem 6 can be utilized to construct a quantum circuit that performs multiplication over the field $GF(2^n)$ with linear depth by using $\mathcal{O}(n \log_2(n))$ ancillary qubits.

To parallelize the circuit, we begin by using n ancillary qubits to store the register $\boldsymbol{c}' = Q^{-1}\boldsymbol{c}$:

$$|\boldsymbol{a}\rangle |\boldsymbol{b}\rangle |\boldsymbol{c}\rangle |\boldsymbol{0}\rangle \mapsto |\boldsymbol{a}\rangle |\boldsymbol{b}\rangle |\boldsymbol{c}\rangle |\boldsymbol{c}'\rangle . \tag{5.27}$$

Then, to apply the recursive formula of Theorem 6, we perform the following transformation:

$$|\boldsymbol{a}\rangle |\boldsymbol{b}\rangle |\boldsymbol{c}\rangle |\boldsymbol{c}'\rangle |\boldsymbol{0}\rangle \mapsto |\boldsymbol{a}_L \oplus \boldsymbol{a}_R\rangle |\boldsymbol{a}_R\rangle |\boldsymbol{b}_L \oplus \boldsymbol{b}_R\rangle |\boldsymbol{b}_R\rangle |\boldsymbol{c}_L\rangle |\boldsymbol{c}_R\rangle |\boldsymbol{c}'_L\rangle |\boldsymbol{c}'_L \oplus \boldsymbol{c}'_R\rangle |\boldsymbol{c}'_L \oplus \boldsymbol{c}_R\rangle . \tag{5.28}$$

This transformation can be done with a CNOT circuit of depth 2. The first layer consists of adding $\boldsymbol{a}_R$ and $\boldsymbol{b}_R$ to $\boldsymbol{a}_L$ and $\boldsymbol{b}_L$ respectively, as well as adding $\boldsymbol{c}'_L$ to $\boldsymbol{c}'_R$ and $\boldsymbol{c}_R$ to the clean ancillae register. The second layer of CNOT gates simply consists of adding $\boldsymbol{c}'_L$ to the last register storing $\boldsymbol{c}_R$. This preparation enables the parallel execution of the first two recursive calls of the formula presented in Theorem 6:

$$g(\boldsymbol{a}_L \oplus \boldsymbol{a}_R, \boldsymbol{b}_L \oplus \boldsymbol{b}_R, \boldsymbol{c}_R) \oplus h(\boldsymbol{a}_L \oplus \boldsymbol{a}_R, \boldsymbol{b}_L \oplus \boldsymbol{b}_R, \boldsymbol{c}'_L) \tag{5.29}$$

and

$$g(\boldsymbol{a}_R, \boldsymbol{b}_R, \boldsymbol{c}'_L \oplus \boldsymbol{c}_R) \oplus h(\boldsymbol{a}_R, \boldsymbol{b}_R, \boldsymbol{c}'_L \oplus \boldsymbol{c}'_R). \tag{5.30}$$

These recursive calls can be executed simultaneously as they operate on independent registers. After these recursive calls, we reverse the transformation of Equation 5.28 using the inverse of the CNOT circuit, which is also done in constant depth. Finally, we realize the following transformation:

$$|\boldsymbol{a}\rangle |\boldsymbol{b}\rangle |\boldsymbol{c}\rangle |\boldsymbol{c}'\rangle |\boldsymbol{0}\rangle \mapsto |\boldsymbol{a}_L\rangle |\boldsymbol{a}_R\rangle |\boldsymbol{b}_L\rangle |\boldsymbol{b}_R\rangle |\boldsymbol{c}_L \oplus \boldsymbol{c}_R\rangle |\boldsymbol{c}_R\rangle |\boldsymbol{c}'_L \oplus \boldsymbol{c}_R\rangle |\boldsymbol{c}'_R\rangle \tag{5.31}$$

This can also be accomplished with a CNOT circuit of depth 2 by first adding $\boldsymbol{c}_R$ to $\boldsymbol{c}_L$ and then adding $\boldsymbol{c}_R$ to $\boldsymbol{c}'_L$. This transformation allows for the execution of the third recursive call from Theorem 6:

$$g(\boldsymbol{a}_L, \boldsymbol{b}_L, \boldsymbol{c}_L \oplus \boldsymbol{c}_R) \oplus h(\boldsymbol{a}_L, \boldsymbol{b}_L, \boldsymbol{c}'_L \oplus \boldsymbol{c}_R). \tag{5.32}$$

An illustrative example of the circuit generated by this procedure for $GF(2^4)$ is provided in Figure 5.1.

Complexity analysis. The transformation described in Equation 5.27 can be implemented by CNOT circuit with a depth of $\mathcal{O}(n)$ [103]. Then, the formula of Theorem 6 is applied to divide the sum of g and h into three recursive calls. Two of these recursive calls are done in parallel by using $\lceil k/2 \rceil$ ancillary qubits, where k is the size of the registers on which g and h are acting. Each recursive call results in either a constant-depth CNOT circuit or a single CCZ gate for the terminal case. Moreover, each recursive call divides the size of the registers by 2. Therefore, the depth of the circuit implementing all these recursive calls is equal to $\mathcal{O}(2^{\log_2(n)}) = \mathcal{O}(n)$. Thus, the total depth of the circuit is $\mathcal{O}(n)$ and the number of ancillary qubits in the circuit is $\mathcal{O}(n \log_2(n))$.

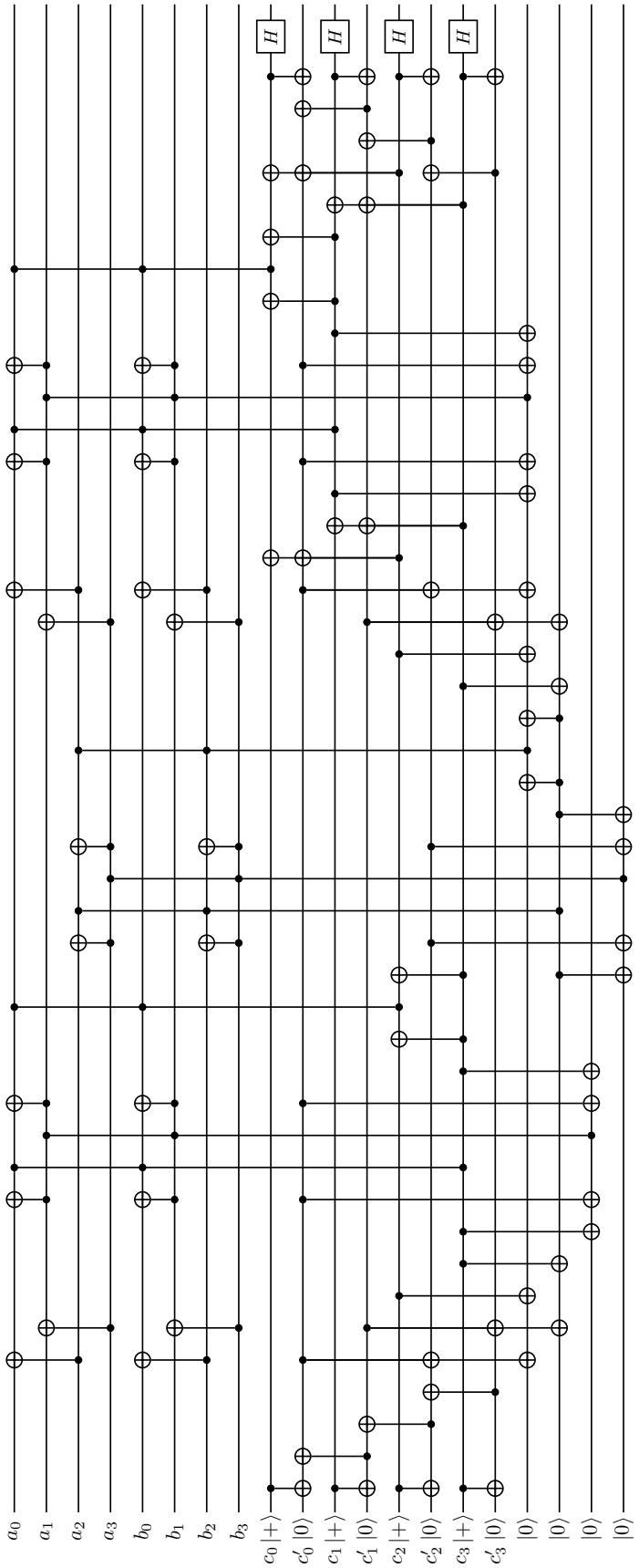


Figure 5.1: Linear depth multiplication circuit for $GF(2^4)$ with primitive polynomial $P(x) = x^4 + x + 1$.

5.3 Logarithmic depth multiplication for particular families of primitive polynomials

In this section, we show how multiplication over $GF(2^n)$ can be done with logarithmic depth in the cases where the primitive polynomial is a trinomial or an equally spaced polynomial.

5.3.1 Trinomials

A primitive trinomial of degree n satisfies:

$$P(x) = x^n + x^k + 1 \tag{5.33}$$

where k is an integer satisfying $1 < k < n$. Let i be an integer such that $0 \leq i < (n-k)$, and let j be an integer such that $0 \leq i + j(n-k) \leq n-2$. The $(i + j(n-k))$-th column of the associated reduction matrix Q represents the coefficients of the polynomial

$$x^{k+i+j(n-k)} + x^{i+j(n-k)} \pmod{P(x)}. \tag{5.34}$$

For $j = 0$, Equation 5.34 simplifies to

$$x^{k+i} + x^i \pmod{P(x)}. \tag{5.35}$$

The CNOT circuit associated with these $(n-k)$ columns can be constructed by applying parallel CNOT gates with the qubit i as control and the qubit $i + k$ as target.

For $j > 0$, we have the following recursive relation:

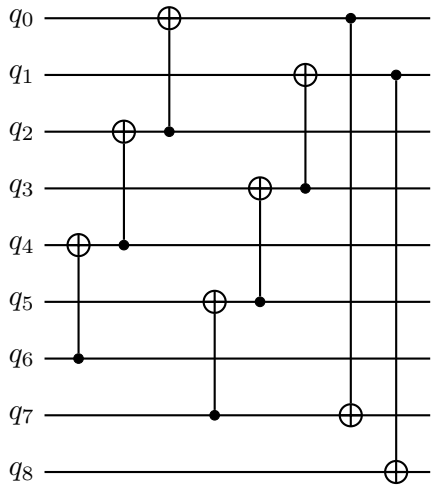
$$\begin{aligned} & x^{k+i+j(n-k)} + x^{i+j(n-k)} \pmod{P(x)} \\ &= x^n x^{i+(j-1)(n-k)} + x^{i+j(n-k)} \pmod{P(x)} \\ &= (x^k + 1)x^{i+(j-1)(n-k)} + x^{i+j(n-k)} \pmod{P(x)} \\ &= x^{k+i+(j-1)(n-k)} + x^{i+(j-1)(n-k)} + x^{i+j(n-k)} \pmod{P(x)}. \end{aligned} \tag{5.36}$$

Therefore, the $(i + j(n-k))$-th column of Q is equal to the $(i + (j-1)(n-k))$-th column of Q plus an additional 1 on the diagonal (corresponding to the term $x^{i+j(n-k)}$). The CNOT circuit associated with these columns can then be constructed by applying $(n-k)$ ladders of CNOT gates in parallel. As proven in Section 4.1, a CNOT ladder over n qubits can be implemented with a depth of $\mathcal{O}(\log_2(n))$. Thus, the CNOT circuit associated with the reduction matrix Q for the primitive trinomial $P(x)$ can be constructed with logarithmic depth.

For example, for the trinomial $P(x) = x^9 + x^7 + 1$, the associated reduction matrix Q is

$$Q = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}. \tag{5.37}$$

A CNOT circuit associated with this reduction matrix is:



where the two ladders of CNOT gates (acting on qubits (q_0, q_2, q_4, q_6) and qubits (q_1, q_3, q_5, q_7)) can be implemented in logarithmic depth by using the circuit produced by Algorithm 1.

5.3.2 Equally spaced polynomials

An equally spaced primitive polynomial $P(x)$ of degree nk satisfies

$$P(x) = \sum_{i=0}^{n} x^{ik}. \tag{5.38}$$

where k is an integer satisfying $0 < k < n$. Let j be an integer such that $0 \leq i < k$. The j-th column of the associated reduction matrix Q represents the coefficients of the polynomial

$$\sum_{i=0}^{n-1} x^{ik+j} \pmod{P(x)}. \tag{5.39}$$

The CNOT circuit associated with these k columns can be constructed by applying k parallel ladders of CNOT gates on the qubits $(j, k+j, 2k+j, \ldots, (n-1)k+j)$.

For all the other columns $j + \ell k$, where ℓ is an integer satisfying $0 < \ell < n$, are representing the coefficients of the polynomial

$$\begin{aligned}
\sum_{i=0}^{n-1} x^{ik+j+\ell k} \pmod{P(x)} &= \sum_{i=0}^{n-1} x^{(i+\ell)k+j} \pmod{P(x)} \\
&= x^{(n+\ell-1)k+j} + \sum_{i=1}^{n-1} x^{(i+\ell-1)k+j} \pmod{P(x)} \\
&= x^{nk} x^{(\ell-1)k+j} + x^{(\ell-1)k+j} \sum_{i=1}^{n-1} x^{ik} \pmod{P(x)} \\
&= x^{(\ell-1)k+j} \left(\sum_{i=0}^{n-1} x^{ik} + \sum_{i=1}^{n-1} x^{ik} \right) \pmod{P(x)} \\
&= x^{(\ell-1)k+j} \pmod{P(x)}.
\end{aligned} \tag{5.40}$$

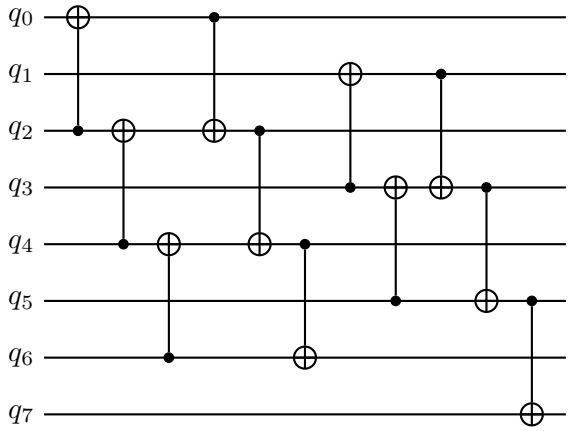
The CNOT circuit associated with these $n(k-1)-1$ columns can be constructed by applying k reversed ladder of CNOT gates on the qubits $(j, k+j, 2k+j, \ldots, (n-1)k+j)$. As proven

in Section 4.1, a CNOT ladder over n qubits can be implemented with a depth of $\mathcal{O}(\log_2(n))$. Thus, the CNOT circuit associated with the reduction matrix Q for the primitive equally spaced polynomial $P(x)$ can be constructed with logarithmic depth.

For example, for $n = 4$ and $k = 2$, we have the equally spaced polynomial $P(x) = x^8 + x^6 + x^4 + x^2 + 1$. Its associated reduction matrix Q is

$$Q = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{5.41}$$

A CNOT circuit associated with this reduction matrix is:



where the $2k$ ladders of CNOT gates can be implemented in logarithmic depth by using the circuit produced by Algorithm 1.

Part III

Quantum Circuit Optimization

Chapter 6

Optimal Hadamard Gate Count for Clifford+R_Z Synthesis of Pauli Rotation Sequences

Fault-tolerant quantum computing enables reliable and large-scale quantum computation at the cost of an important resource overhead when compared to an error-free model. Much work has been put into quantum circuit optimization in order to reduce this additional cost and make fault-tolerant quantum computing more practical and scalable. In particular, numerous algorithms have been designed to minimize the number of T gates in a quantum circuit [21, 104–114]. This focus on T-count minimization is primarily due to the sizable amount of resources, in terms of time and number of qubits, generally required by fault-tolerance protocols, such as magic state distillation [38], to implement the T gate. In contrast, Clifford operations can typically be implemented at little expense in most common quantum error correcting codes via transversal operations, code deformation [59] or lattice surgery [37]. In such context, and considering the fact that the Clifford+T gate set is approximatively universal, the T-count stands out as a key metric to minimize in order to make fault-tolerant quantum computing more efficient. Moreover, minimizing the T-count is also crucial in the field of quantum circuits simulation as many simulators have a runtime that scales exponentially with respect to the number of T gates [115–119].

The problem of finding the optimal number of T gates in a $\{\text{CNOT}, S, T\}$ circuit composed of n qubits has been well formalized for $\{\text{CNOT}, S, T\}$ circuits by demonstrating its equivalence with the problem of finding a maximum likelihood decoder for the punctured Reed-Muller code of order $n-4$ and length 2^n-1 [106], which is tantamount to the third order symmetric tensor rank decomposition problem [120]. In order to make use of this formalism in Clifford+T circuits it is necessary to circumvent the Hadamard gates in some way; this can be achieved by applying one of the two following strategies. The first method consists of extracting $\{\text{CNOT}, S, T\}$ subcircuits and interposing them with layers of Hadamard gates [104]. Then an independent and Hadamard-free instance of the T-count minimization problem can be formulated for each $\{\text{CNOT}, S, T\}$ subcircuit extracted. The second strategy involves a measurement-based gadget which can substitute a Hadamard gate. This Hadamard gadgetization procedure requires the following additional resources for each Hadamard gate gadgetized: an ancilla qubit, a CZ gate and a measurement [121].

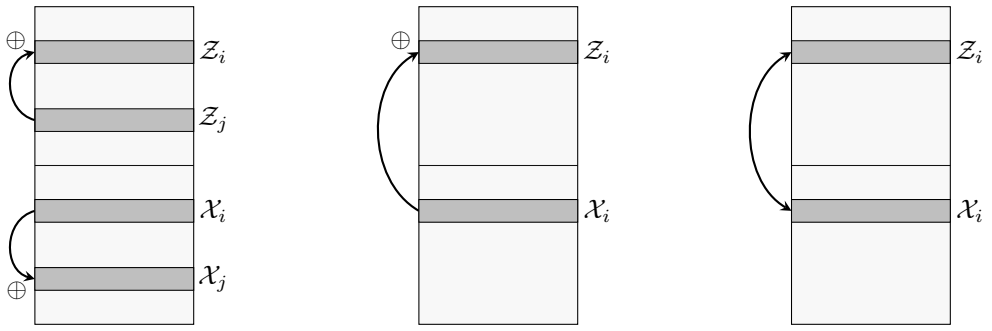
The number of T gates in a circuit containing h Hadamard gates can be upper bounded by $\mathcal{O}(n^2h)$ or $\mathcal{O}((n+h)^2)$ in the case where all Hadamard gates are gadgetized [106]. Hence,

each Hadamard gate that must be circumvented, regardless of the strategy applied, for a lack of a good Hadamard gate optimization procedure is potentially the cause of missed opportunities for further T gate reduction. Therefore, a preliminary procedure consisting in reducing the number of Hadamard gates can result in an important T-count reduction, as demonstrated in Reference [105]. It has been shown that circumventing all Hadamard gates using the Hadamard gadgetization procedure is the strategy that leads to the best reduction in the number of T gates [108]. However, the main drawback of this method is the use of one additional qubit for each Hadamard gate gadgetized. This is obviously an inconvenience if the number of qubits at disposal is limited, but can also be detrimental to the optimization process in two ways. Firstly, as suggested in Reference [112], it may become more difficult to find opportunities to reduce the T-count as the ratio between the number of qubits and the number of T gates increases. In addition, the runtime of a T-count optimizer can drastically increase as the number of qubits grows. For all these reasons it is important to minimize the number of auxiliary qubits needed, which further motivates investigations into a pre-processing step optimizing the number Hadamard gates in the initial circuit.

We can mainly distinguish two strategies for the optimization of quantum circuits. The first one is referred to as pattern matching and involves the detection of patterns of gates within the circuit to then substitute them by an equivalent, but nonetheless different, sequence of gates. A series of transformation is therefore applied to the circuit, but its semantic is preserved at each step of the process. This method has already been applied to the optimization of Hadamard gates by using rewriting rules that preserve or reduce the number of Hadamard gates within the circuit [105, 112]. The second method is circuit re-synthesis which consists in extracting some parts of the circuit, representing them by higher level constructs and performing their synthesis to obtain an equivalent circuit. This method has not yet been considered for the optimization of Hadamard gates, despite displaying excellent performances for other optimization problems such as T gate reduction [108, 110].

In the case of circuit re-synthesis, a commonly used fact is that the operation performed by a given Clifford+T circuit can be represented by a sequence of $\pi/4$ Pauli rotations followed by a final Clifford operator, as explained in Section 1.2.3.4. A strategy for optimizing the number of Hadamard gates could then consist of synthesizing this sequence of $\pi/4$ Pauli rotations using as few Hadamard gates as possible. In this chapter, we present a polynomial-time algorithm that optimally solves this problem. With the Hadamard gadgetization approach, a Hadamard gate needs to be gadgetized only if it comes after and precedes a T gate in the circuit, we say that such Hadamard gates are internal Hadamard gates. This leads to a more specific Hadamard gate reduction problem consisting in reducing the number of internal Hadamard gates within the circuit. We demonstrate that our Hadamard gate optimization algorithm can be adapted to also solve this problem optimally and in polynomial time as well. Our algorithms are not working exclusively for the Clifford+T gate set but can also be executed on any circuit composed of $\{X, \text{CNOT}, S, H, R_Z\}$ gates.

Chapter organization. The problems tackled in this chapter are presented and formalized in Section 6.1. In Section 6.2, we introduce an optimal algorithm for synthesizing a sequence of Pauli rotation with a minimal number of Hadamard gates. On the basis of this result, in Section 6.3, we present an algorithm for synthesizing a sequence of Pauli rotation with a minimal number of internal Hadamard gates. Section 6.4 presents alternative versions of our algorithms with lower computational complexities. Finally, benchmarks are provided in Section 6.5 to evaluate the performances and scalability of our algorithms on a library of reversible logic circuits and on large-scale quantum circuits.



(a) $\text{CNOT}_{i,j}$ (b) S_i (c) H_i

Figure 6.1: Operations on a sequence of Pauli products $\mathcal{S} = \begin{bmatrix} \mathcal{Z} \\ \mathcal{X} \end{bmatrix}$ corresponding to the conjugation of all its Pauli products by a Clifford gate. For the $\text{CNOT}_{i,j}$ gate where i is the control qubit and j is the target qubit (a), the $\mathcal{Z}_j$ and $\mathcal{X}_i$ rows are added to the $\mathcal{Z}_i$ and $\mathcal{X}_j$ rows respectively. For a S gate applied on qubit i (b), the $\mathcal{X}_i$ row is added to the $\mathcal{Z}_i$ row. For a H gate applied on qubit i (c), the $\mathcal{Z}_i$ and $\mathcal{X}_i$ rows are swapped.

6.1 Problem statement

6.1.1 Pauli rotation sequences

A Pauli operator $P \in \mathcal{P}_n^*$ can be encoded using $2n+1$ bits: $2n$ bits for the n Pauli matrices and 1 bit for the sign [17]. In the following we will encode a Pauli operator $P \in \mathcal{P}_n^*$ with $2n$ bits and neglect its sign as it has no impact on the formulation of our problem; we will use the term Pauli product to designate a Pauli operator deprived of its sign. Let $\mathcal{S} = \begin{bmatrix} \mathcal{Z} \\ \mathcal{X} \end{bmatrix}$ be a block matrix of size $2n \times m$ representing a sequence of m Pauli products acting on n qubits such that $\mathcal{Z}$ is the submatrix of $\mathcal{S}$ formed by its first n rows and $\mathcal{X}$ is the submatrix of $\mathcal{S}$ formed by its last n rows. The value $(\mathcal{Z}_{i,j}, \mathcal{X}_{i,j})$ represents the ith component of the jth Pauli product encoded by $\mathcal{S}$, such that the values $(0,0), (0,1), (1,1)$ and $(1,0)$ are corresponding to the Pauli matrices I, X, Y and Z respectively. We use the notation $\mathcal{S}_{:,i}$ to refer to the column i of the matrix $\mathcal{S}$, we will denote $P(\mathcal{S}_{:,i})$ the Pauli product encoded by $\mathcal{S}_{:,i}$, and we will say that $\mathcal{S}_{:,i}$ is diagonal if and only if $P(\mathcal{S}_{:,i})$ is diagonal and that $\mathcal{S}_{:,i}$ and $\mathcal{S}_{:,j}$ commute (or anticommute) if their associated Pauli products $P(\mathcal{S}_{:,i})$ and $P(\mathcal{S}_{:,j})$ commute (or anticommute). Throughout the document we use the zero-based indexing for vectors and matrices and the initial element is termed the zeroth element, for instance the zeroth column of $\mathcal{S}$ is $\mathcal{S}_{:,0}$. If all Pauli products encoded by $\mathcal{S}$ are conjugated by a Clifford gate $U \in \{\text{CNOT}, S, H\}$, then $\mathcal{S}$ can be updated to encode the Pauli products $U^\dagger P(\mathcal{S}_{:,i})U$, for all i, via the operations depicted in Figure 6.1. These operations are analogous to the operations performed in the tableau representation [17], presented in Section 1.2.1.2. We will say that $\tilde{\mathcal{S}} = U^\dagger \mathcal{S} U$ if and only if $P(\tilde{\mathcal{S}}_{:,i}) = \pm U^\dagger P(\mathcal{S}_{:,i})U$ for all i and for some Clifford operator U.

As explained in Section 1.2.3.4, any Clifford+R_Z circuit can be represented up to a global phase by a sequence of Pauli rotations followed by a final Clifford operator [21]. The synthesis of a Pauli rotation is then a key procedure for constructing an equivalent Clifford+R_Z circuit from this representation. Let U be a Clifford operator such that

$$U^\dagger R_P(\theta) U = R_{U^\dagger P U}(\theta) = R_{Z_i}(\theta) \tag{6.1}$$

for some qubit i and some Pauli operator $P \in \mathcal{P}_n^*$. Then the synthesis of the Pauli rotation $R_P(\theta)$ can be performed by implementing U, $U^\dagger$ and inserting a $R_Z(\theta)$ gate in between on qubit i. If P is diagonal then the Clifford operator U satisfying Equation 6.1 can be implemented using only CNOT and X gates. Otherwise, if P is not diagonal, at least one Hadamard gate is required to implement U over the $\{X, \text{CNOT}, S, H\}$ gate set such that

$$U^\dagger R_P(\theta) U = R_{P'}(\theta) \tag{6.2}$$

where P' is diagonal. Note that the gate set considered is not minimal as the X gate can be generated from the S and H gates. As our cost model is the number of Hadamard gates we include the X gate so that no H gates are required to implement it. The X gate finds its purpose in the case where

$$U^\dagger R_P(\theta) U = R_{U^\dagger P U}(\theta) = R_{Z_i}(-\theta) \tag{6.3}$$

by allowing the implementation of the $R_Z(-\theta)$ gate using the $R_Z(\theta)$ gate via the equality

$$R_Z(-\theta) = X R_Z(\theta) X. \tag{6.4}$$

Nonetheless, in the case where $\theta = \pi/4$, the minimal $\{\text{CNOT}, S, H\}$ gate set can be used since the negative sign can be compensated by inserting three S gates as

$$U R_{Z_i}(7\pi/4) U^\dagger = U R_{Z_i}(-\pi/4) U^\dagger = R_{U Z_i U^\dagger}(-\pi/4) = R_P(\pi/4). \tag{6.5}$$

In this chapter, unless indicated otherwise, the term Clifford circuit will refer to a circuit exclusively composed of gates from the set $\{X, \text{CNOT}, S, H\}$, the use of other Clifford gate sets is discussed at the end of Section 6.2.2.

6.1.2 Diagonalization network

The synthesis of a sequence of Pauli rotations using the Clifford+R_Z gate set implies the construction of a diagonalization network, derived from the notion of parity network established in Reference [122] and which is defined as follows.

Definition 1 (Diagonalization network)**.** A Clifford circuit C is a diagonalization network for a sequence $\mathcal{S}$ of m Pauli products if and only if there exists m non-negative integers $\alpha_0 \leq \ldots \leq \alpha_{m-1}$ such that $U_i^\dagger P(\mathcal{S}_{:,i}) U_i$ is diagonal, where U_i is the Clifford operator implemented by the first α_i gates of C.

A sequence of m Pauli rotations can be represented by a triple $(\mathcal{S}, \boldsymbol{b}, \boldsymbol{\theta})$, where $\mathcal{S}$ encodes a sequence of m Pauli products, $\boldsymbol{b} \in \{-1, 1\}^m$ and $\boldsymbol{\theta} \in \mathbb{R}^m$ such that b_i and θ_i correspond to the sign and angle associated with the Pauli product $P(\mathcal{S}_{:,i})$. Let C be a diagonalization network for $\mathcal{S}$, then the sequence of Pauli rotations represented by $(\mathcal{S}, \boldsymbol{b}, \boldsymbol{\theta})$ can be easily implemented from C up to a final Clifford circuit by inserting m $\{X, \text{CNOT}, R_Z\}$ subcircuits into C. Indeed, as stated previously, if a Pauli product P is diagonal then the Clifford operator V satisfying

$$V^\dagger R_P(\theta) V = R_{V^\dagger P V}(\theta) = R_{Z_j}(\theta) \tag{6.6}$$

for some qubit j, can be implemented using only CNOT and X gates. And because C is a diagonalization network for $\mathcal{S}$ then by definition there exists m non-negative integers $\alpha_0 \leq \ldots \leq \alpha_{m-1}$ such that $U_i^\dagger P(\mathcal{S}_{:,i}) U_i$ is diagonal, where U_i is the Clifford operator implemented by the first

α_i gates of C. It follows that inserting, for all i and just after the α_ith gate of C, a $\{\text{CNOT}, X\}$ implementation of the Clifford operators V_i and $V_i^\dagger$ with the $R_{Z_j}(b_i\theta_i)$ gate in between, such that V_i satisfies

$$V_i^\dagger U_i^\dagger P(\mathcal{S}_{:,i}) U_i V_i = R_{Z_j}(b_i\theta_i) \tag{6.7}$$

for some qubit j, will result in an implementation of the sequence of Pauli rotations defined by $(\mathcal{S}, \boldsymbol{b}, \boldsymbol{\theta})$ up to a final Clifford circuit.

The circuit obtained by this procedure obviously contains the same number of Hadamard gates as C as no additional Hadamard gate was inserted. Thus, synthesizing a sequence of Pauli rotations represented by $(\mathcal{S}, \boldsymbol{b}, \boldsymbol{\theta})$ with a minimal number of Hadamard gates up to a final Clifford operator is equivalent to the problem of constructing a diagonalization network for $\mathcal{S}$ using a minimal number of Hadamard gates. This approach can easily be extended to take into account the final Clifford operator, as explained in Section 6.2.3. We define $h(C)$ as being the number of Hadamard gates in a Clifford circuit C, and we extend the notation for a sequence of Pauli products $\mathcal{S}$ such that $h(\mathcal{S}) = \min\{h(C) \mid C \text{ is a diagonalization network for } \mathcal{S}\}$. The problem of synthesizing a sequence of Pauli rotations ignoring the final Clifford operator with a minimal number of Hadamard gates can then be defined as follows.

Problem 1 (H-Opt)**.** *Given a sequence $\mathcal{S}$ of Pauli products, find a Clifford circuit C that is a diagonalization network for $\mathcal{S}$ and such that $h(C) = h(\mathcal{S})$.*

In Clifford+T circuits, the Hadamard gadgetization procedure aims to transform the circuit in order to obtain an Hadamard-free subcircuit containing all the T gates. Hence, a Hadamard gate does not need to be gadgetized if there is no T gate preceding it. To take this particularity into consideration we define the following problem relating to the synthesis of a sequence of Pauli rotations up to a final Clifford circuit with a minimal number of internal Hadamard gates.

Problem 2 (Internal-H-Opt)**.** *Given a sequence $\mathcal{S}$ of Pauli products, find a Clifford circuit $C = C_1 :: C_2$, i.e. C is the circuit resulting from the concatenation of C_1 and C_2, such that $h(C_2)$ is minimized and C_2 is a diagonalization network for $\tilde{\mathcal{S}} = U^\dagger \mathcal{S} U$ where U is the Clifford operator associated with C_1.*

In Section 6.2.1, we propose a diagonalization network synthesis algorithm to solve the H-Opt problem. We prove its optimality in Section 6.2.2, and it is then employed in Section 6.3 to design an algorithm solving the Internal-H-Opt problem.

6.2 Hadamard gate minimization

6.2.1 Diagonalization network synthesis algorithm

We first describe a simple procedure, of fundamental importance in our diagonalization network synthesis algorithm, to construct a Clifford operator U such that $U^\dagger P U$ is diagonal, where P is a non-diagonal Pauli product. Let i be such that $P_i \in \{X, Y\}$, which necessarily exists as P is non-diagonal. If there exists $j \neq i$ such that $P_j \in \{X, Y\}$, then, based on the operation depicted in Figure 6.1(a), we can deduce that the Pauli product P' resulting from the conjugation of P by the $\text{CNOT}_{i,j}$ gate satisfies $P'_i \in \{X, Y\}$, $P'_j \in \{I, Z\}$ and $P'_k = P_k$ for all $k \notin \{i, j\}$. More generally, if $P' = U^\dagger P U$ where U is the Clifford operator associated with the fan-out formed by the gates $\{\text{CNOT}_{i,j} \mid P_j \in \{X, Y\}, \forall j \neq i\}$, then P'_j is diagonal for all $j \neq i$. To complete the diagonalization of P' we then just have to make P'_i diagonal while preserving this property. If $P'_i = Y$ then conjugating P' by a S gate on qubit i maps P'_i to X. And in the case where

Algorithm 5: Diagonalization network synthesis with a minimal number of H gates

Input: A sequence $\mathcal{S} = \begin{bmatrix} \mathcal{Z} \\ \mathcal{X} \end{bmatrix}$ of m Pauli products.

Output: A diagonalization network for $\mathcal{S}$ with a minimal number of H gates.

procedure `DiagonalizationNetworkSynthesis(`$\mathcal{S}$`)`
  $C \leftarrow$ new empty circuit
  if $\mathcal{S}$ is empty **then**
    return C
  end
  if $\exists i$ such that $\mathcal{X}_{i,0} = 1$ **then**
    foreach $j \in \{j \mid \mathcal{X}_{j,0} = 1\} \setminus \{i\}$ **do**
      $C \leftarrow C :: \mathrm{CNOT}_{i,j}$
      $\mathcal{S} \leftarrow \mathrm{CNOT}_{i,j}\, \mathcal{S}\, \mathrm{CNOT}_{i,j}$
    end
    if $\mathcal{Z}_{i,0} = 1$ **then**
      $C \leftarrow C :: S_i$
      $\mathcal{S} \leftarrow S_i\, \mathcal{S}\, S_i$
    end
    $C \leftarrow C :: H_i$
    $\mathcal{S} \leftarrow H_i\, \mathcal{S}\, H_i$
  end
  $\mathcal{S} \leftarrow \mathcal{S}$ with its first column removed
  return $C ::$ `DiagonalizationNetworkSynthesis(`$\mathcal{S}$`)`

$P'_i = X$, then conjugating P' by a H gate on qubit i maps P'_i to Z, and our diagonalization procedure is complete as the S_i and H_i operations do not affect P'_j where $j \neq i$.

Consider the diagonalization network synthesis algorithm whose pseudo-code is given in Algorithm 5 and which takes a sequence $\mathcal{S}$ of m Pauli products as input. The algorithm constructs a Clifford circuit C iteratively by processing the Pauli products constituting $\mathcal{S}$ in order. When a Pauli product $P = P(\mathcal{S}_{:,i})$ is being processed, if $U^\dagger P U$, where $U \in \mathcal{C}_n$ is the Clifford operator implemented by C, is diagonal then the algorithm moves on to the next Pauli product. Otherwise, if $U^\dagger P U$ is not diagonal, a sequence of gates, constructed using the procedure described above, are appended to C so that the updated Pauli product $U^\dagger P U$ is diagonal. Thus, Algorithm 5 outputs a Clifford circuit that is a diagonalization network for $\mathcal{S}$. A detailed execution example of Algorithm 5 is provided in Section 6.2.1.3.

6.2.1.1 Complexity analysis

At each iteration Algorithm 5 carries out at most $\mathcal{O}(n)$ row operations on $\mathcal{S}$ where n is the number of qubits, m iterations are performed and $\mathcal{S}$ has m columns, therefore the complexity of Algorithm 5 is $\mathcal{O}(nm^2)$.

In the typical case where $n < m$, a faster version of Algorithm 5 can be implemented using the tableau representation [17]. Let $\mathcal{T}$ be a tableau initialized at the begining of the algorithm. Instead of updating $\mathcal{S}$ for each Clifford gate appended to the circuit C, we can use $\mathcal{T}$ to keep track of the Clifford operator U implemented by C. For each Clifford gate appended to C, $\mathcal{T}$ can be updated in $\mathcal{O}(n)$ [17]. Then, the algorithm proceeds in the same way as Algorithm 5 by

sequentially diagonalizing the Pauli products represented by $\mathcal{S}$. However, the ith Pauli product to diagonalized is not $P(\mathcal{S}_{:,i})$ but $U^\dagger P(\mathcal{S}_{:,i})U$, which can be computed in $\mathcal{O}(n^2)$ using the tableau $\mathcal{T}$. This operation must be performed $\mathcal{O}(m)$ times and $\mathcal{T}$ must be updated $\mathcal{O}(nm)$ times as the number of gates in the final Clifford circuit is $\mathcal{O}(nm)$, therefore the overall time complexity of this algorithm is $\mathcal{O}(n^2m)$. More details on this approach are given in Section 6.4, where this algorithm is adapated to take a Clifford+R_Z circuit as input instead of a sequence of Pauli products.

6.2.1.2 Hadamard gate count

In order to evaluate $h(C)$, where C is the output circuit of Algorithm 5, we will rely on the following definition.

Definition 2 (Commutativity matrix)**.** Let $\mathcal{S}$ be a sequence of m Pauli products. The commutativity matrix $A^{(\mathcal{S})}$ associated with $\mathcal{S}$ is a strictly upper triangular Boolean matrix of size $m \times m$ such that for all $i < j$:

$$A^{(\mathcal{S})}_{i,j} = \begin{cases} 0 & \text{if } \mathcal{S}_{:,i} \text{ commutes with } \mathcal{S}_{:,j}, \\ 1 & \text{if } \mathcal{S}_{:,i} \text{ anticommutes with } \mathcal{S}_{:,j}. \end{cases}$$

For convenience we will drop the superscript $(\mathcal{S})$ from A when it is clear from the context that A is associated with $\mathcal{S}$. The commutativity matrix $A^{(\mathcal{S})}$ can also be seen as the adjacency matrix of a directed acyclic graph, which has already been studied and linked to the T-depth optimization problem [110]. In this work, we further reinforce the interest in this structure by establishing a relation between the H-Opt and Internal-H-Opt problems and the rank of $A^{(\mathcal{S})}$. Note that if $\tilde{\mathcal{S}} = U^\dagger \mathcal{S} U$, where U is some Clifford operator, then $A^{(\tilde{\mathcal{S}})} = A^{(\mathcal{S})}$ because if two Pauli products P and P' are commuting (or anticommuting) then $U^\dagger P U$ and $U^\dagger P' U$ are commuting (or anticommuting).

The number of Hadamard gates in the circuit produced by Algorithm 5 can be characterized via the following theorem.

Theorem 8. *Let* $\mathcal{S} = \begin{bmatrix} \mathcal{Z} \\ \mathcal{X} \end{bmatrix}$ *be a sequence of* m *Pauli products,* A *be its commutativity matrix and* C *be the Clifford circuit returned by Algorithm 5 when* $\mathcal{S}$ *is given as input. Then* $h(C) = \operatorname{rank}(M)$ *where* $M = \begin{bmatrix} \mathcal{X} \\ A \end{bmatrix}$.

Proof. Let $\mathcal{S}^{(i)} = \begin{bmatrix} \mathcal{Z}^{(i)} \\ \mathcal{X}^{(i)} \end{bmatrix}$ be the sequence of Pauli products given as input to the ith recursive call of Algorithm 5 with $\mathcal{S}^{(0)} = \mathcal{S}$, and let $M^{(i)} = \begin{bmatrix} \mathcal{X}^{(i)} \\ A^{(i)} \end{bmatrix}$ where $A^{(i)}$ is the commutativity matrix associated with $\mathcal{S}^{(i)}$. We first start by analyzing how $M^{(i)}$ evolves to $M^{(i+1)}$ when $\mathcal{S}^{(i)}_{:,0}$ is diagonal. In such case, we can obtain $M^{(i+1)}$ from $M^{(i)}$ by removing the first column of $M^{(i)}$ and the first row of its submatrix $A^{(i)}$. Let $P = P(\mathcal{S}^{(i)}_{:,0})$, then, because P is diagonal, the following equation holds:

$$\bigoplus_{k \in K} \mathcal{X}^{(i)}_k = \bigoplus_{k \in K} M^{(i)}_k = A^{(i)}_0 \tag{6.8}$$

where $K = \{k \mid \mathcal{Z}^{(i)}_{k,0} = 1\}$. Indeed, as P is diagonal we necessarily have $P_k = Z$ for some k, and in the case where $P_j = I$ for all $j \neq k$ we have $\mathcal{X}^{(i)}_{k,j} = 1$ if and only if $\mathcal{S}^{(i)}_{:,0}$ anticommutes with

$\mathcal{S}^{(i)}_{:,j}$, and so $\mathcal{X}^{(i)}_k = A^{(i)}_0$. In a more general case, if there exists $j \neq k$ satisfying $P_j = Z$ then we can apply a $\text{CNOT}_{j,k}$ gate for all such j in order to fall back on our previous case, which implies Equation 6.8. Consequently, removing the first row of the submatrix $A^{(i)}$ will not change the rank of $M^{(i)}$. Moreover, due to the fact that $\mathcal{S}^{(i)}_{:,0}$ is diagonal, the first column of $M^{(i)}$ is equal to the null vector. Therefore we have $\text{rank}(M^{(i+1)}) = \text{rank}(M^{(i)})$ when $\mathcal{S}^{(i)}_{:,0}$ is diagonal.

In the case where $\mathcal{S}^{(i)}_{:,0}$ is not diagonal, Algorithm 5 will apply a sequence of CNOT and S gates followed by a single H gate. Let $\tilde{\mathcal{S}}^{(i)} = \begin{bmatrix} \tilde{\mathcal{Z}}^{(i)} \\ \tilde{\mathcal{X}}^{(i)} \end{bmatrix}$ be the sequence of Pauli products obtained by conjugating all Pauli products of $\mathcal{S}^{(i)}$ by this sequence of CNOT and S gates, and let $\tilde{M}^{(i)} = \begin{bmatrix} \tilde{\mathcal{X}}^{(i)} \\ A^{(i)} \end{bmatrix}$. Note that we have $\text{rank}(\tilde{M}^{(i)}) = \text{rank}(M^{(i)})$ as applying a S or CNOT operation on $\mathcal{S}^{(i)}$ does not change the rank of $\mathcal{X}^{(i)}$. Let j be the qubit on which the Hadamard gate is applied, we must have $\tilde{M}^{(i)}_{j,0} = 1$ and $\tilde{M}^{(i)}_{k,0} = 0$ for all $k \neq j$, which implies that $\tilde{M}^{(i)}_j$ is independent from all the other rows of $\tilde{M}^{(i)}$. Let $\hat{M}^{(i)} = \begin{bmatrix} \hat{\mathcal{X}}^{(i)} \\ A^{(i)} \end{bmatrix}$ where $\hat{\mathcal{S}}^{(i)} = \begin{bmatrix} \hat{\mathcal{Z}}^{(i)} \\ \hat{\mathcal{X}}^{(i)} \end{bmatrix}$ is obtained by conjugating all Pauli products of $\tilde{\mathcal{S}}^{(i)}$ by a Hadamard gate on qubit j, and notice that $\hat{M}^{(i)}_k = \tilde{M}^{(i)}_k$ for all $k \neq j$. Analogously to Equation 6.8, since $\hat{\mathcal{S}}^{(i)}_{:,0}$ is diagonal, the following equation holds:

$$\bigoplus_{k \in \hat{K}} \hat{\mathcal{X}}^{(i)}_k = \bigoplus_{k \in \hat{K}} \hat{M}^{(i)}_k = A^{(i)}_0 \tag{6.9}$$

where $\hat{K} = \{k \mid \hat{\mathcal{Z}}^{(i)}_{k,0} = 1\}$. Furthermore, as $j \in \hat{K}$, $\hat{M}^{(i)}_j$ can be expressed as follows:

$$\hat{M}^{(i)}_j = \bigoplus_{k \in \hat{K} \setminus \{j\}} \hat{M}^{(i)}_k \oplus A^{(i)}_0 = \bigoplus_{k \in \tilde{K}} \tilde{M}^{(i)}_k \oplus A^{(i)}_0 \tag{6.10}$$

where $\tilde{K} = \{k \mid \tilde{\mathcal{Z}}^{(i)}_{k,0} = 1\} = \hat{K} \setminus \{j\}$. It follows that $\hat{M}^{(i)}_j$ is a linear combination of the rows of $\tilde{M}^{(i)}$ whereas $\tilde{M}^{(i)}_j$ is an independent row, and so $\text{rank}(\hat{M}^{(i)}) = \text{rank}(\tilde{M}^{(i)}) - 1$. After the Hadamard gate has been applied we end up in the same case as when $\mathcal{S}^{(i)}_{:,0}$ is diagonal, therefore we have $\text{rank}(M^{(i+1)}) = \text{rank}(\hat{M}^{(i)}) = \text{rank}(M^{(i)}) - 1$.

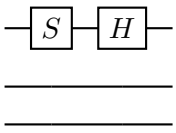
We demonstrated that $\text{rank}(M^{(i+1)}) = \text{rank}(M^{(i)})$ when no Hadamard gate is applied at the ith recursive call, and that $\text{rank}(M^{(i+1)}) = \text{rank}(M^{(i)}) - 1$ if one Hadamard gate is applied. Thus, the number of Hadamard gates in the Clifford circuit C is equal to $\text{rank}(M) - \text{rank}(M^{(m)})$ where m is the number of Pauli products in $\mathcal{S}$. The sequence of Pauli products $\mathcal{S}^{(m)}$ is empty, hence $\text{rank}(M^{(m)}) = 0$ and $h(C) = \text{rank}(M)$. □

6.2.1.3 Diagonalization network synthesis example

In this section, we provide a detailed execution example of Algorithm 5 which performs the synthesis of a diagonalization network for a given sequence of Pauli products. Let $\mathcal{S}$ be the sequence of Pauli products given as input to Algorithm 5 and defined as follows:

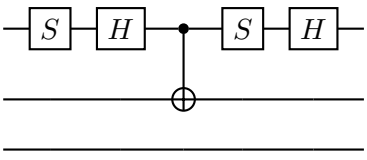
$$\mathcal{S} = \begin{bmatrix} \mathcal{Z} \\ \mathcal{X} \end{bmatrix} = \left(\begin{array}{cccc} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ \hline 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{array}\right).$$

The algorithm starts by diagonalizing the Pauli product represented by the first column of $\mathcal{S}$. This is done by inserting a S gate in the circuit followed by a Hadamard gate on the first qubit. The matrix $\mathcal{S}$ encoding the sequence of Pauli products is updated by performing the operations associated with the S and H gates, as depicted in Figure 6.1. Then, the first column is removed from the matrix and the algorithm performs a recursive call on the updated matrix.



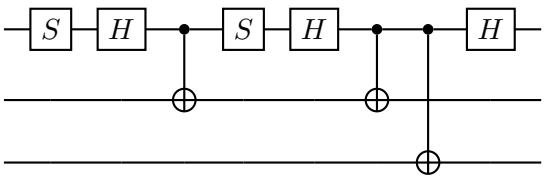
$$\left(\begin{array}{ccc} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ \hline 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{array}\right)$$

This time, the first column of the lower matrix has a Hamming weight greater than one. Therefore, the algorithm inserts a CNOT gate acting on the first and second qubits of the circuit to reduce the Hamming weight of the first column of the lower matrix to one. The Pauli product encoded by the first column can then be diagonalized by inserting a S gate and a Hadamard gate on the first qubit. Then, the matrix is updated, the first column is removed from the matrix and the algorithm performs a recursive call.



$$\left(\begin{array}{cc} 0 & 1 \\ 0 & 1 \\ 0 & 0 \\ \hline 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{array}\right)$$

Again, the first column of the lower matrix has a Hamming weight greater than one. This time two CNOT gates must be inserted in the circuit to reduce it to one. After that, a Hadamard gate is inserted to diagonalize the Pauli product encoded by the first column.

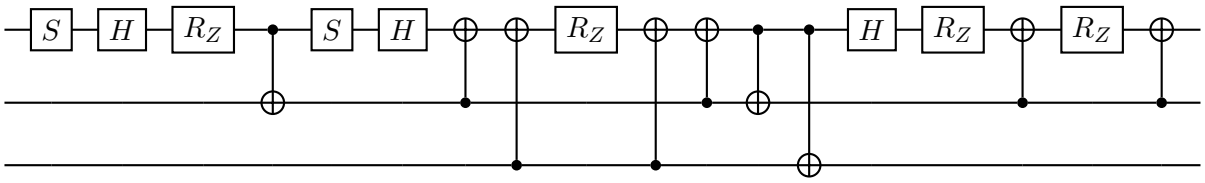


$$\left(\begin{array}{c} 1 \\ 1 \\ 0 \\ \hline 0 \\ 0 \\ 0 \end{array}\right)$$

Finally, the Pauli product encoded by the remaining column is already diagonal. Therefore, the algorithm simply removes the column from the matrix. The matrix is then empty so the

algorithm terminates by returning the constructed circuit, which is a diagonalization network for the sequence of Pauli products encoded by $\mathcal{S}$.

We can then insert $\{\text{CNOT}, R_Z\}$ subcircuits in the appropriate places to implement the sequence of Pauli rotations associated with $\mathcal{S}$ up to a final Clifford circuit. The following figure shows an example of a possible circuit obtained after this procedure.



The commutativity matrix $A^{(\mathcal{S})}$ associated with $\mathcal{S}$ is

$$A^{(\mathcal{S})} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

As stated by Theorem 8, we can notice that the number of Hadamard gates in the circuit produced by Algorithm 5 is equal to $\text{rank}(M)$ where $M = \begin{bmatrix} \mathcal{X} \\ A^{(\mathcal{S})} \end{bmatrix}$.

6.2.2 Optimality

In this section, we demonstrate the optimality of Algorithm 5 by proving the following theorem.

Theorem 9. *Let $\mathcal{S} = \begin{bmatrix} \mathcal{Z} \\ \mathcal{X} \end{bmatrix}$ be a sequence of m Pauli products, A be its commutativity matrix and C be a Clifford circuit that optimally solves the H-Opt problem for $\mathcal{S}$. Then $h(C) = \text{rank}(M)$ where $M = \begin{bmatrix} \mathcal{X} \\ A \end{bmatrix}$.*

Our proof of Theorem 9 rests on the following proposition, which puts an upper bound on the number of Hadamard gates required to simultaneously diagonalize a set of mutually commuting Pauli products.

Proposition 1. *Let $\mathcal{S} = \begin{bmatrix} \mathcal{Z} \\ \mathcal{X} \end{bmatrix}$ be a sequence of m mutually commuting Pauli products of size n and let $U \in \mathcal{C}_n$ be a Clifford operator such that $U^\dagger P(\mathcal{S}_{:,i}) U$ is diagonal for all i. Then $h(C) \geq \text{rank}(\mathcal{X})$, where C is a Clifford circuit implementing U.*

Proof. Let $\mathcal{S}^{(i)}$ be the state of $\mathcal{S}$ resulting from conjugating all its Pauli products by the Clifford operator implemented by the first i gates of C. If the $(i+1)$th gate of C is a CNOT or S gate, then $\text{rank}(\mathcal{X}^{(i+1)}) = \text{rank}(\mathcal{X}^{(i)})$. Else, if the $(i+1)$th gate of C is a Hadamard gate, then $\mathcal{X}^{(i+1)}$ and $\mathcal{X}^{(i)}$ have at least $n-1$ rows in common and $1 \geq |\,\text{rank}(\mathcal{X}^{(i)}) - \text{rank}(\mathcal{X}^{(i+1)})| \geq 0$. Therefore, the number of Hadamard gates in C is at least $|\,\text{rank}(\mathcal{X}) - \text{rank}(\mathcal{X}^{(k)})|$, where k is the number of gates in C. The circuit C performs a simultaneous diagonalization of the Pauli products constituting $\mathcal{S}$, which implies that $\text{rank}(\mathcal{X}^{(k)}) = 0$, hence $h(C) \geq |\,\text{rank}(\mathcal{X}) - \text{rank}(\mathcal{X}^{(k)})| = \text{rank}(\mathcal{X})$.

□

In the following we use $\mathcal{S}_{:,:j}$ to denote the submatrix formed by the first j columns of $\mathcal{S}$. Theorem 8 implies an upper bound on the number of Hadamard gates required to solve the H-Opt problem. There always exists a Clifford circuit C that is a diagonalization network for a sequence of Pauli products $\mathcal{S} = \begin{bmatrix}\mathcal{Z}\\ \mathcal{X}\end{bmatrix}$ such that $\text{rank}(M) \geq h(C)$ where $M = \begin{bmatrix}\mathcal{X}\\ A\end{bmatrix}$ and A is the commutativity matrix associated with $\mathcal{S}$. In order to prove Theorem 9 it remains to show that if C is a diagonalization network for $\mathcal{S}$ then $h(C) \geq \text{rank}(M)$. To do so, we will show that we can derive a Clifford circuit C' from C such that C' is satisfying $h(C') = h(C)$ and is a solution to a specific instance $\mathcal{S}'$ of the simultaneous diagonalization problem, where $\mathcal{S}' = \begin{bmatrix}\mathbf{0}\\ M'\end{bmatrix}$ is a sequence of mutually commuting Pauli products satisfying $\text{rank}(M') \geq \text{rank}(M)$. By Proposition 1 we then would have $h(C) = h(C') \geq \text{rank}(M') \geq \text{rank}(M)$. We first give a construction for M' and prove that $\text{rank}(M') \geq \text{rank}(M)$ via the following proposition.

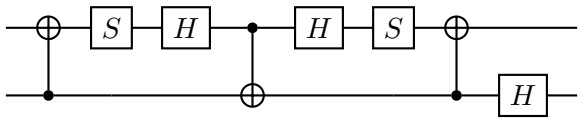
Proposition 2. *Let C be a diagonalization network for a sequence $\mathcal{S} = \begin{bmatrix}\mathcal{Z}\\ \mathcal{X}\end{bmatrix}$ of m Pauli products of size n, and let $C^{(i)}$ be the subcircuit of C truncated after its ith Hadamard gate. And let the matrices $\mathcal{S}^{(i)} = \begin{bmatrix}\mathcal{Z}^{(i)}\\ \mathcal{X}^{(i)}\end{bmatrix}$ be such that $\mathcal{S}^{(0)} = \mathcal{S}$ and in the case where $i > 0$:*

$$\begin{aligned} \mathcal{S}^{(i)}_{:,j} &= \mathbf{0} && \text{if } C^{(i-1)} \text{ is a diagonalization network for } \mathcal{S}_{:,:j},\\ P(\mathcal{S}^{(i)}_{:,j}) &= \pm U^{\dagger}_{(i)} P(\mathcal{S}_{:,j}) U_{(i)} && \text{otherwise,} \end{aligned}$$

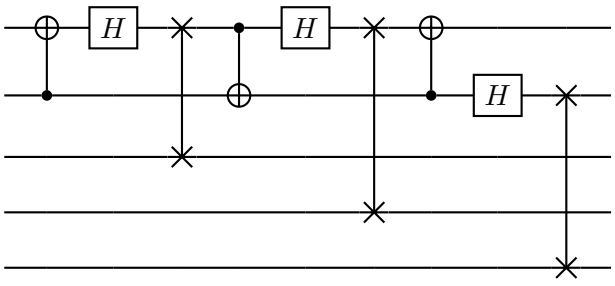
where $U_{(i)} \in \mathcal{C}_n$ is the Clifford operator associated with $C^{(i)}$. Consider the matrices $M = \begin{bmatrix}\mathcal{X}\\ A\end{bmatrix}$ and $M' = \begin{bmatrix}\mathcal{X}\\ A'\end{bmatrix}$ where A is the commutativity matrix associated with $\mathcal{S}$, and A' is a matrix composed of $h(C)$ rows such that $A'_{i-1} = \mathcal{X}^{(i)}_j$ where j is the qubit on which the ith Hadamard gate of C is applied. Then we have $\text{rank}(M') \geq \text{rank}(M)$.

Proof. We will prove Proposition 2 by showing that the rows of M are in the span of the set formed by the rows of the $\mathcal{X}^{(i)}$ matrices, which are themselves in the row space of the matrix M'. We first show that the rows of $\mathcal{X}^{(i)}$ are in the span of the set formed by the first $n+i$ rows of M'. For $i = 0$, this assertion is obviously true as we have $\mathcal{X}_j = M'_j$ for all $0 \leq j < n$. Let $\tilde{\mathcal{S}}^{(i)} = \begin{bmatrix}\tilde{\mathcal{Z}}^{(i)}\\ \tilde{\mathcal{X}}^{(i)}\end{bmatrix} = U^{\dagger}\mathcal{S}^{(i)}U$, where U is the Clifford operator implemented by the $\{\text{CNOT}, S\}$ subcircuit comprised between the ith and $(i+1)$th Hadamard gate of C. Note that performing a CNOT or S operation on $\mathcal{S}^{(i)}$ doesn't change the row space of $\mathcal{X}^{(i)}$, therefore the rows of $\tilde{\mathcal{X}}^{(i)}$ are in the row space of $\mathcal{X}^{(i)}$. Let $\boldsymbol{\alpha}$ be a vector of size m such that α_j is the smallest non-negative integer for which $C^{(\alpha_j)}$ is a diagonalization network for $\mathcal{S}_{:,:j}$ and let k be the qubit on which the $(i+1)$th Hadamard gate of C is applied. We can then distinguish 3 cases for the value of α_j, where $0 \leq j < m$:

- $\alpha_j < i$, then $\mathcal{S}^{(i+1)}_{:,j} = \tilde{\mathcal{S}}^{(i)}_{:,j} = \mathbf{0}$ for all j, because if $C^{(i-1)}$ is a diagonalization network for $\mathcal{S}_{:,:j}$ then $C^{(i)}$ is also a diagonalization network for $\mathcal{S}_{:,:j}$;
- $\alpha_j = i$, then $\mathcal{S}^{(i+1)}_{:,j} = \mathbf{0}$ and $\tilde{\mathcal{S}}^{(i)}_{:,j}$ is diagonal, therefore $\mathcal{X}^{(i+1)}_{:,j} = \tilde{\mathcal{X}}^{(i)}_{:,j} = \mathbf{0}$;



(a) Circuit C that is a diagonalization network for $\mathcal{S}$ where $\mathcal{S}_{:,0}$ is diagonalized by the first 3 gates, $\mathcal{S}_{:,1}$ by the first 5 gates and $\mathcal{S}_{:,2}$ by the whole circuit.



(b) Circuit C' derived from C.

$$\mathcal{S} = \begin{bmatrix} \mathcal{Z} \\ \mathcal{X} \end{bmatrix} = \left(\begin{array}{ccc} 1 & 0 & 0 \\ 1 & 1 & 0 \\ \hline 1 & 0 & 1 \\ 0 & 1 & 0 \end{array}\right), \quad A^{(\mathcal{S})} = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

(c) Initial sequence of Pauli products and its commutativity matrix.

$$\mathcal{S}' = \begin{bmatrix} \mathbf{0} \\ M' \end{bmatrix} = \left(\begin{array}{ccc} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \hline 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{array}\right)$$

(d) Pauli products that are simultaneously diagonalized by C', where M' is as defined in Proposition 2.

Figure 6.5: Construction example of C' and $\mathcal{S}'$ for the proof of Theorem 9.

- $\alpha_j > i$ then $\tilde{\mathcal{S}}^{(i)}_{:,j}$ encodes the same Pauli product as $\mathcal{S}^{(i+1)}_{:,j}$ up to a Hadamard operation on qubit k, therefore $\mathcal{X}^{(i+1)}_{\ell,j} = \tilde{\mathcal{X}}^{(i)}_{\ell,j}$ for all $\ell \neq k$.

To sum up we have $\mathcal{X}^{(i+1)}_{j} = \tilde{\mathcal{X}}^{(i)}_{j}$ for all $j \neq k$ and by definition we have $\mathcal{X}^{(i+1)}_{k} = M'_{n+i+1}$. It follows that the rows of $\mathcal{X}^{(i+1)}$ are in the span of the set formed by the first $n+i+1$ rows of M'. We now show that, for all j, A_j is in the row space of $\mathcal{X}^{(\alpha_j)}$. Since $\mathcal{S}^{(\alpha_j)}_{:,j}$ is diagonal, similarly to Equation 6.8, the following holds:

$$\bigoplus_{k \in K} \mathcal{X}^{(\alpha_j)}_{k,\ell} = \begin{cases} 0 & \text{if } \mathcal{S}_{:,\ell} \text{ commutes with } \mathcal{S}_{:,j} \text{ or } \ell \leq j, \\ 1 & \text{if } \mathcal{S}_{:,\ell} \text{ anticommutes with } \mathcal{S}_{:,j}, \forall \ell > j, \end{cases} \tag{6.11}$$

where $K = \{k \mid \mathcal{Z}^{(\alpha_j)}_{k,j} = 1\}$. This sum satisfy the same properties as the row j of the commutativity matrix A associated with $\mathcal{S}$, therefore we have:

$$A_j = \bigoplus_{k \in K} \mathcal{X}^{(\alpha_j)}_{k}. \tag{6.12}$$

Thus, for all j, A_j is in the row space of the matrix $\mathcal{X}^{(\alpha_j)}$, whose rows are themselves in the row space of M'. Consequently, M_j is in the row space of M' for all j and $\text{rank}(M') \geq \text{rank}(M)$.

□

The proof of Theorem 9 can now be formulated based on Proposition 1 and 2.

Proof of Theorem 9. Consider a Clifford C' containing the same number of Hadamard gates as C, acting over $n + h(C)$ qubits and constructed by the following process:

1. Start with C' as a copy of C with $h(C)$ additional qubits.
2. Remove all S gates from C'.
3. After the ith Hadamard gate of C', insert a SWAP gate operating over the qubits $n+i-2$ and j where j is the qubit on which the ith Hadamard gate is applied.

An example of this process is provided in Figure 6.5. The $\text{SWAP}_{i,j}$ gate can be implemented using CNOT gates:

$$\text{SWAP}_{i,j} = \text{CNOT}_{i,j}\text{CNOT}_{j,i}\text{CNOT}_{i,j}$$

This operation can be performed on $\mathcal{S}$ by swapping the rows $\mathcal{Z}_i$ and $\mathcal{Z}_j$ as well as the rows $\mathcal{X}_i$ and $\mathcal{X}_j$. Let M' be defined as in Proposition 2, let $\mathcal{S}' = \begin{bmatrix} \mathbf{0} \\ M' \end{bmatrix}$ be a sequence of m mutually commuting Pauli products of size $n + h(C)$ and let U' is the Clifford operator implemented by C', we will show that $U'^{\dagger} P(\mathcal{S}'_{:,i}) U'$ is diagonal for all i. We reuse $C^{(i)}$ and $\mathcal{S}^{(i)} = \begin{bmatrix} \mathcal{Z}^{(i)} \\ \mathcal{X}^{(i)} \end{bmatrix}$ as defined in Proposition 2, and we define $\mathcal{S}'^{(i)} = \begin{bmatrix} \mathcal{Z}'^{(i)} \\ \mathcal{X}'^{(i)} \end{bmatrix}$ and $C'^{(i)}$ analogously where $C'^{(i)}$ is the subcircuit resulting from truncating C' after its ith inserted SWAP gate and $C'^{(0)}$ is the empty circuit.

We now prove by induction that for all $0 \leq i \leq h(C)$, $0 \leq j < n$ and $n \leq k < n + i$ we have $\mathcal{X}'^{(i)}_j = \mathcal{X}^{(i)}_j$, $\mathcal{Z}'^{(i)}_j = \mathbf{0}$ and $\mathcal{X}'^{(i)}_k = \mathbf{0}$. For $i = 0$ and $0 \leq j < n$, the equalities $\mathcal{X}'_j = \mathcal{X}_j$ and $\mathcal{Z}'^{(i)}_j = \mathbf{0}$ are satisfied by definition. Let $0 \leq i < h(C)$ and α_i be the qubit on which the $(i+1)$th Hadamard gate of C is applied. The matrix $\mathcal{S}^{(i+1)}$ can be obtained from $\mathcal{S}^{(i)}$ by performing a sequence of $\{\text{CNOT}, S\}$ operations and a Hadamard operation on qubit α_i. Similarly, the matrix $\mathcal{S}'^{(i+1)}$ can be obtained from $\mathcal{S}'^{(i)}$ by performing the same sequence of CNOT operations, a Hadamard operation on qubit α_i and a SWAP operation acting on the qubits α_i and $n + i$. In both cases, the rows $\mathcal{X}^{(i)}_j$ and $\mathcal{X}'^{(i)}_j$, where $0 \leq j < n, j \neq \alpha_i$, are only affected by the CNOT operations, and so if $\mathcal{X}'^{(i)}_j = \mathcal{X}^{(i)}_j$ for all $0 \leq j < n$, then $\mathcal{X}'^{(i+1)}_j = \mathcal{X}^{(i+1)}_j$ for all $0 \leq j < n, j \neq \alpha_i$. Notice that the only gate in C' acting on the qubit $n + i$ is the SWAP gate operating on the qubits α_i and $n+i$ and recall that by definition $\mathcal{X}'_{n+i} = \mathcal{X}^{(i+1)}_{\alpha_i}$; then, because this SWAP gate is the last gate of the circuit $C'^{(i+1)}$, we have $\mathcal{X}'^{(i+1)}_{\alpha_i} = \mathcal{X}'_{n+i} = \mathcal{X}^{(i+1)}_{\alpha_i}$. Therefore, for all $0 \leq j < n$, if $\mathcal{X}'^{(i)}_j = \mathcal{X}^{(i)}_j$ then $\mathcal{X}'^{(i+1)}_j = \mathcal{X}^{(i+1)}_j$.

If $\mathcal{Z}'^{(i)}_j = \mathbf{0}$ for all $0 \leq j < n$, then applying a sequence of CNOT operations on $\mathcal{S}'^{(i)}$ acting on the first n qubits will not alter the matrix $\mathcal{Z}'^{(i)}$. Thus, if $\mathcal{Z}'^{(i)}_j = \mathbf{0}$ for all $0 \leq j < n$, then $\mathcal{Z}'^{(i+1)}_j = \mathcal{Z}'^{(i)}_j = \mathbf{0}$ for all $0 \leq j < n, j \neq \alpha_i$. Furthermore, if $\mathcal{Z}'^{(i)}_{\alpha_i} = \mathbf{0}$ for all $0 \leq j < n$, then applying a Hadamard operation on $\mathcal{S}'^{(i)}$ acting on qubit α_i after this sequence of CNOT operations would yield $\mathcal{X}'^{(i)}_{\alpha_i} = \mathbf{0}$. This Hadamard operation is followed by a SWAP operation between the qubits α_i and $k = n + i$ which would induce that $\mathcal{X}'^{(i+1)}_k = \mathbf{0}$ and $\mathcal{Z}'^{(i+1)}_{\alpha_i} = \mathbf{0}$ because $\mathcal{Z}'_k = \mathbf{0}$. Thus, for all $0 \leq j < n$, if $\mathcal{Z}'^{(i)}_j = \mathbf{0}$ then $\mathcal{Z}'^{(i+1)}_j = \mathbf{0}$. In addition, for all k such that $n \leq k < n + i$, the circuit $C'^{(i+1)}$ doesn't contain any gate operating on the qubit k other than those included in $C'^{(i)}$; therefore if $\mathcal{X}'^{(i)}_k = \mathbf{0}$ for all $n \leq k < n + i$ then $\mathcal{X}'^{(i+1)}_k = \mathbf{0}$ for all $n \leq k < n + i + 1$.

Let $i = h(C)$, by combining the facts that $\mathcal{X}'^{(i)}_j = \mathcal{X}^{(i)}_j = \mathbf{0}$ for all $0 \leq j < n$ and $\mathcal{X}'^{(i)}_{n+j} = \mathbf{0}$ for all $0 \leq j < i$, we can deduce that $\mathcal{X}'^{(i)}$ is the null matrix which imply that $U'^{\dagger} P(\mathcal{S}'_{:,j}) U'$ is

diagonal for all j where U' is the Clifford operator implemented by C'. By Proposition 1 we have $h(C) = h(C') \geq \text{rank}(M')$, and by Proposition 2 we have $\text{rank}(M') \geq \text{rank}(M)$ which entails $h(C) \geq \text{rank}(M)$. This lower bound is satisfied by Algorithm 5 as stated by Theorem 8, this implies that Algorithm 5 is optimal and concludes the proof of Theorem 9. □

6.2.2.1 Pauli rotations ordering

Algorithm 5 solves the H-Opt problem for a fixed sequence of Pauli rotations. However, if two adjacent Pauli rotations are commuting then their order could be inverted, leading to another sequence of Pauli rotations representing the same operator. We show that changing the ordering in this way doesn't affect the minimal number of Hadamard gate required to implement the diagonalization network associated with the sequence of Pauli rotations. Let $\mathcal{S} = \begin{bmatrix} \mathcal{Z} \\ \mathcal{X} \end{bmatrix}$ be a sequence of Pauli products, let i be such that $\mathcal{S}_{:,i}$ commutes with $\mathcal{S}_{:,i+1}$ and let $\mathcal{S}' = \begin{bmatrix} \mathcal{Z}' \\ \mathcal{X}' \end{bmatrix}$ be a sequence of Pauli products obtained by swapping the columns i and $i+1$ of $\mathcal{S}$. Let $M = \begin{bmatrix} \mathcal{X} \\ A \end{bmatrix}$ and $M' = \begin{bmatrix} \mathcal{X}' \\ A' \end{bmatrix}$ where A and A' are the commutativity matrices of $\mathcal{S}$ and $\mathcal{S}'$ respectively. Since $\mathcal{S}_{:,i}$ commutes with $\mathcal{S}_{:,i+1}$ we have $A_{i,i+1} = A'_{i,i+1} = 0$, and so $M_{:,i} = M'_{:,i+1}$ and $M_{:,i+1} = M'_{:,i}$. The matrix M' can be obtained from M by swapping its columns i and $i+1$, which entails $\text{rank}(M) = \text{rank}(M')$. Thus, inverting the order of two adjacent and commuting Pauli rotations doesn't change the minimal number of Hadamard gates required to implement the diagonalization network associated with the sequence of Pauli rotations.

6.2.2.2 Other gate sets

One could consider the problem over other Clifford gate sets, which raises the question of whether these gate sets could perform better than the $\{X, \text{CNOT}, S, H\}$ gate set considered. In order to achieve a number of Hadamard gate less than $\text{rank}(M)$, where M is defined as in Theorem 9, the gate set considered needs to have at least one gate, other than the Hadamard gate, such that its decomposition over the $\{X, \text{CNOT}, S, H\}$ gate set necessarily involves at least one Hadamard gate. Said otherwise, the number of Hadamard gates is at least $\text{rank}(M)$ for any gate set in which the Hadamard gate is the only gate U for which there exists a non-diagonal Pauli operator P such that $U^\dagger PU$ is diagonal.

6.2.3 Extension to Clifford+R_Z circuit re-synthesis

Any Clifford+R_Z circuit can be characterized by a sequence of Pauli rotations followed by a final Clifford operator C_f [21]. We demonstrated that Algorithm 5 solves the H-Opt problem optimally, and so it can be used to synthesize a sequence of Pauli rotations up to a final Clifford operator $C_{f'}$ with a minimal number of Hadamard gates. The synthesis of the full Clifford+R_Z circuit can then be performed by coupling Algorithm 5 with a procedure to synthesize the Clifford operator $C_f \cdot C_{f'}$. We will demonstrate that this procedure can in fact also be performed by Algorithm 5 with a minimal number of Hadamard gates.

A Clifford operator $U \in \mathcal{C}_n$ can be represented by a tableau encoding $2n$ Pauli operators such that n of them are mutually commuting Pauli operators called stabilizer generators and the other half are also mutually commuting Pauli operators referred to as destabilizer generators. If

the stabilizer generators are all diagonalized, then the Clifford operator can be synthesized using only $\{X, S, \text{CNOT}\}$ gates [17]. Thus, synthesizing a Clifford operator with the minimal number of Hadamard gates amounts to finding a Clifford circuit C containing the minimal number of Hadamard gates and such that $U^\dagger PU$ is diagonal for all P in the stabilizer generators, where U is the Clifford operator associated with C. We will demonstrate via the following proposition that a Clifford circuit satisfying these properties is produced by Algorithm 5 when the sequence of Pauli products $\mathcal{S}$ given as input encodes the stabilizer generators on any order.

Proposition 3. *Let $\mathcal{S}$ be a sequence of m mutually commuting Pauli products and C be the Clifford circuit returned by Algorithm 5 when $\mathcal{S}$ is given as input. Then $U^\dagger P(\mathcal{S}_{:,j})U$ is diagonal for all j, where U is the Clifford operator associated with C.*

Proof. Let P and P' be commuting Pauli operators such that P is diagonal and P' is not diagonal. If there exists k such that $P'_k = X$ and $P'_\ell \in \{I, Z\}$ for all $\ell \neq k$, then $P_k = I$ because P commutes with P' and P is diagonal. Therefore conjugating P and P' with a Hadamard gate on qubit k will result in both operators being diagonalized. Let $C^{(i)}$ be the subcircuit of C truncated before its ith Hadamard gate with $C^{(0)}$ defined as the empty circuit, and let $U_{(i)}$ be the Clifford operator associated with $C^{(i)}$. Due to the construction process of C, for each subcircuit $C^{(i)}$ where $i > 0$ there exists j such that $P' = U^\dagger_{(i)} \mathcal{S}_{:,j} U_{(i)}$ satisfies $P'_k = X$ and $P'_\ell \in \{I, Z\}$ for all $\ell \neq k$ where k is the qubit on which the ith Hadamard gate of C is applied. Hence, for all $i < h(C)$ and for all j, if $U^\dagger_{(i)} P(\mathcal{S}_{:,j}) U_{(i)}$ is diagonal, then $U^\dagger_{(i+1)} P(\mathcal{S}_{:,j}) U_{(i+1)}$ is also diagonal. The circuit C is a diagonalization network for $\mathcal{S}$, which imply that for all j there exists $U_{(i)}$ such that $U^\dagger_{(i)} P(\mathcal{S}_{:,j}) U_{(i)}$ is diagonal, and so $U^\dagger P(\mathcal{S}_{:,j})U$ is a diagonal for all j.

□

Based on Proposition 3, we can now show that Algorithm 5 can be used to synthesize a sequence of Pauli rotations followed by a final Clifford operator with a minimal number of Hadamard gates. Let $\mathcal{S}$ be a sequence of Pauli products associated with the sequence of Pauli rotations we are aiming to implement, let $\mathcal{S}'$ be a sequence of Pauli products encoding the stabilizer generators of the final Clifford operator, and let $\tilde{\mathcal{S}} = \begin{bmatrix} \mathcal{S} & \mathcal{S}' \end{bmatrix}$. Any $\{X, \text{CNOT}, S, H, R_Z\}$ circuit implementing this sequence of Pauli rotations followed by the final Clifford operator is necessarily a diagonalization network for $\tilde{\mathcal{S}}$. The circuit C returned by Algorithm 5 when $\tilde{\mathcal{S}}$ is given as input satisfies this condition with a minimal number of Hadamard gates. Moreover, as indicated by Proposition 3, C simultaneously diagonalize the sequence of Pauli products encoded by $\mathcal{S}'$. Thus, the synthesis of the sequence of Pauli rotations and the final Clifford operator can be completed with a minimal number of Hadamard gate by inserting $\{X, \text{CNOT}, S, R_Z\}$ subcircuits into C. Moreover, the minimal number of Hadamard gate is equal to $\text{rank}(M)$, which satisfies $\text{rank}(A) \leq \text{rank}(M) \leq \text{rank}(A) + n$, where n is the number of qubits, A is the commutativity matrix, and M is the extended commutativity. The extended commutativity M is defined as follows, where a commutativity matrix $A^{(\mathcal{S})}$ is said to be associated a Clifford+R_Z circuit C if $\mathcal{S}$ is associated with the sequence of Pauli rotations implemented by C up to a Clifford circuit.

Definition 3 (Extended commutativity matrix)**.** Let C be a Clifford+R_Z circuit, and let C' be a Clifford+R_Z circuit constructed from C by inserting an $R_Z(\theta)$ gate, with θ not being a multiple of $\pi/2$, at the beginning and at the end of the circuit on each qubit. The extended commutativity matrix M associated with C is the commutativity matrix associated with C'.

Let C be the circuit obtained once the number of Hadamard gates have been optimized with our method. The circuit C may contain an important number of CNOT gates as our algorithm

does not aim at optimizing the CNOT-count. If necessary, several methods can be used to optimize the number of CNOT gates in C while preserving the number of Hadamard gates. First, the Clifford parts of C can be re-synthesized by using a Clifford circuit synthesis algorithm that preserve the optimal number of Hadamard gates. For example, as shown in Reference [123], a Clifford circuit can be implemented with the optimal number of Hadamard gates via two $\{\text{CNOT}, \text{CZ}, S\}$ circuits separated by a layer of Hadamard gates. Algorithms designed for the synthesis of $\{\text{CNOT}, \text{CZ}, S\}$ circuits can then be used to optimize the number of gates or the depth of the circuit [124]. A complementary way of optimizing the number of CNOT gates in C is to re-synthesize the Hadamard-free subcircuits of C via a phase polynomial synthesis algorithm [7], [122]. This method would probably be more effective than the re-synthesis of the Clifford parts approach when C contains large Hadamard-free subcircuits.

6.3 Internal Hadamard gate minimization

In this section, we tackle the problem of minimizing the number of internal Hadamard gates, which corresponds to the number of Hadamard gates occurring between the first and the last non-Clifford R_Z gate of the circuit. We first give an algorithm in Section 6.3.1 that performs the synthesis of a diagonalization network while minimizing the number of internal Hadamard gates. We then prove its optimality in Section 6.3.2.

6.3.1 Algorithm

Solving the Internal-H-Opt problem for a sequence $\mathcal{S} = \begin{bmatrix} \mathcal{Z} \\ \mathcal{X} \end{bmatrix}$ of Pauli products consists in finding a Clifford operator U such that $\text{rank}(\tilde{M})$ is minimal where $\tilde{M} = \begin{bmatrix} \tilde{\mathcal{X}} \\ A \end{bmatrix}$, $\tilde{\mathcal{S}} = \begin{bmatrix} \tilde{\mathcal{Z}} \\ \tilde{\mathcal{X}} \end{bmatrix} = U^\dagger \mathcal{S} U$ and A is the commutativity matrix associated with $\mathcal{S}$. The inequality $\text{rank}(A) \leq \text{rank}(\tilde{M}) \leq \text{rank}(M) \leq \text{rank}(A) + n$, where $M = \begin{bmatrix} \mathcal{X} \\ A \end{bmatrix}$ and n is the number of qubits, imply that the circuit produced by Algorithm 5 contains at most n additional internal Hadamard gates when compared to an optimal solution. To go beyond this approximation and obtain an optimal solution, it is necessary to find a sequence of Clifford operations which, when applied to $\mathcal{S}$, transform $\mathcal{X}$ into $\tilde{\mathcal{X}}$. As discussed in Section 6.2.3, implementing a Clifford operator can be done in two parts: finding a circuit that simultaneously diagonalizes the stabilizer generators of the Clifford operator and finishing the implementation with a $\{X, S, \text{CNOT}\}$ circuit. The $\{X, S, \text{CNOT}\}$ circuit can be disregarded as the associated operations have no impact on the rank of $\tilde{M}$. Hence, solving the Internal-H-Opt problem for a sequence $\mathcal{S}$ of Pauli products consists in finding a set of mutually commuting Pauli products, encoded in a matrix $\mathcal{S}'$, that are simultaneously diagonalized by a Clifford operator U and such that $\text{rank}(\tilde{M})$ is minimal where $\tilde{M} = \begin{bmatrix} \tilde{\mathcal{X}} \\ A \end{bmatrix}$ and $\tilde{\mathcal{S}}$ is the sequence of Pauli products resulting from conjugating all the Pauli products of $\mathcal{S}$ by U. As stated by Proposition 3, a circuit that simultaneously diagonalizes the Pauli products of $\mathcal{S}'$ is produced by Algorithm 5 when $\mathcal{S}'$ is given as input. Thus, if $\begin{bmatrix} \mathcal{S}' & \mathcal{S} \end{bmatrix}$ is given as input to Algorithm 5, then the constructed circuit is a diagonalization network for $\mathcal{S}$ which contains a minimal number of internal Hadamard gates. An example of the execution of Algorithm 6 is given in Figure 6.6.

We propose an algorithm, whose pseudo-code is given in Algorithm 6, to solve the Internal-H-Opt problem optimally by finding the Pauli products constituting $\mathcal{S}'$. Let J_m be an exchange

Algorithm 6: Diagonalization network synthesis with a minimal number of internal H gates

Input: A sequence $\mathcal{S}$ of m Pauli products of size n.
Output: A diagonalization network for $\mathcal{S}$ with a minimal number of internal H gates.

```
J_m ← exchange matrix of size m × m
C ← DiagonalizationNetworkSynthesis(SJ_m)
S' ← stabilizer generators of the inverse of the Clifford operator associated with C
return DiagonalizationNetworkSynthesis([S' S])
```

matrix of size $m \times m$ defined as follows:

$$J_{m_{i,j}} = \begin{cases} 1 & \text{if } i + j = m - 1, \\ 0 & \text{otherwise.} \end{cases} \tag{6.13}$$

As such, the Pauli products encoded by the columns of the matrix $\mathcal{S}J_m$ are then the same as the ones encoded by $\mathcal{S}$ but in reverse order. The algorithm starts by performing a call to Algorithm 5 to obtain a Clifford circuit C that is a diagonalization network for the sequence of Pauli products encoded in $\mathcal{S}J_m$. Then, a set of stabilizer generators associated with the inverse of C are encoded in the columns of $\mathcal{S}'$ and a second and final call to Algorithm 5 is performed where $\left[\mathcal{S}' \quad \mathcal{S}\right]$ is given as input. We prove that the resulting circuit gives an optimal solution to the Internal-H-Opt problem in the next subsection. When one uses Algorithm 6 to perform the re-synthesis of a circuit, as explained in Section 6.2.3, the stabilizer generators associated with the final Clifford operator of the input circuit can be append to the final call to Algorithm 5 to obtain a full re-synthesis of the circuit containing both a minimal number of Hadamard gates and internal Hadamard gates.

Note that a set of stabilizer generators associated with the inverse of the Clifford circuit C can be computed in $\mathcal{O}(n^2 m)$ using the tableau representation as C is composed of $\mathcal{O}(nm)$ gates and a tableau can be updated in $\mathcal{O}(n)$ operations when a Clifford gate is applied. The complexity of the algorithm then resides in the two calls made to Algorithm 5. The first call has a complexity of $\mathcal{O}(n^2 m)$ as $\mathcal{S}J_m$ is composed of m Pauli products. For the second call, $n + m$ Pauli products are given as input because a Clifford operator acting on n qubits has n stabilizer generators. This induces a complexity of $\mathcal{O}(n^2(n + m)) = \mathcal{O}(n^3 + n^2 m)$, which corresponds to $\mathcal{O}(n^2 m)$ in the typical case where $n \leq m$. Thus, the overall complexity of Algorithm 6 matches the complexity of Algorithm 5.

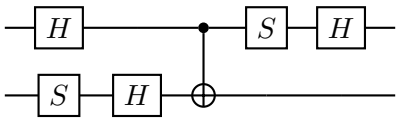
6.3.2 Optimality

This subsection is dedicated to the proof of the following theorem, which states the optimality of Algorithm 6.

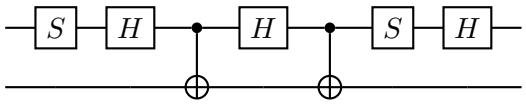
Theorem 10. *Let $\mathcal{S}$ be a sequence of m Pauli products, A be its commutativity matrix and let C be the Clifford circuit returned by Algorithm 6 when $\mathcal{S}$ is given as input. Then C optimally solves the Internal-H-Opt problem with* $\text{rank}(A)$ *internal Hadamard gates.*

We first show that the optimal number of internal Hadamard gates is equal to $\text{rank}(A)$. Our proof rests on the following proposition.

Proposition 4. *Let $\mathcal{S}$ be a sequence of m Pauli products, A be its commutativity matrix and let $\boldsymbol{y}, \boldsymbol{y}'$ be such that $A\boldsymbol{y} = \boldsymbol{0}$ and $A\boldsymbol{y}' = \boldsymbol{0}$. Then the Pauli products encoded by $\mathcal{S}\boldsymbol{y}$ and $\mathcal{S}\boldsymbol{y}'$ are commuting.*



(a) Circuit produced by Algorithm 5 when $\mathcal{S}J_m$ is given as input.



(b) Circuit produced by Algorithm 5 when $\begin{bmatrix}\mathcal{S}' & \mathcal{S}\end{bmatrix}$ is given as input. The first 4 gates are a diagonalization network for $\mathcal{S}'$, the last 3 gates are a diagonalization for $\tilde{\mathcal{S}}$. The whole circuit solves the Internal-H-Opt problem for $\mathcal{S}$ with $\text{rank}(A^{(\mathcal{S})}) = 1$ internal Hadamard gates.

$$\mathcal{S} = \begin{bmatrix}\mathcal{Z}\\ \mathcal{X}\end{bmatrix} = \left(\begin{array}{ccc} 1 & 0 & 0\\ 1 & 1 & 0\\ \hline 1 & 0 & 1\\ 0 & 1 & 0\end{array}\right), \quad A^{(\mathcal{S})} = \begin{pmatrix} 0 & 1 & 1\\ 0 & 0 & 0\\ 0 & 0 & 0\end{pmatrix}$$

(c) Initial sequence of Pauli products and its commutativity matrix, which are the same as the example in Figure 6.5.

$$\mathcal{S}' = \left(\begin{array}{cc} 1 & 0\\ 1 & 1\\ \hline 1 & 1\\ 0 & 1\end{array}\right), \quad \tilde{\mathcal{S}} = \left(\begin{array}{ccc} 0 & 0 & 1\\ 1 & 1 & 0\\ \hline 0 & 1 & 1\\ 0 & 1 & 1\end{array}\right)$$

(d) Sequence of Pauli products $\mathcal{S}'$ and $\tilde{\mathcal{S}}$ such that $\mathcal{S}'$ is associated with the stabilizer generators of the Clifford operator $U^\dagger$ where U is implemented by the circuit depicted in Subfigure 6.6(a), and $\tilde{\mathcal{S}} = U^\dagger \mathcal{S} U$.

Figure 6.6: Example of an execution of Algorithm 6. For a sequence of Pauli products $\mathcal{S}$ (c), the first call to Algorithm 5 will produce a circuit (a) with associated Pauli products $\mathcal{S}'$ (d). The algorithm will then output the circuit produced by Algorithm 5 when $\begin{bmatrix}\mathcal{S}' & \mathcal{S}\end{bmatrix}$ is given as input (b).

Proof. Notice that the Pauli product $\mathcal{S}_{:,i}$ commutes with $\mathcal{S}_{:,j}$ if and only if $(A \oplus A^T)_{i,j} = (A \oplus A^T)_{j,i} = 0$. Then $\mathcal{S}\boldsymbol{y}'$ commutes with $\mathcal{S}_{:,i}$ if and only if $v_i = 0$, where $\boldsymbol{v} = (A \oplus A^T)\boldsymbol{y}'$. And $\mathcal{S}\boldsymbol{y}'$ commutes with $\mathcal{S}\boldsymbol{y}$ if and only if $\boldsymbol{y}^T\boldsymbol{v} = \boldsymbol{y}^T(A \oplus A^T)\boldsymbol{y}' = 0$. As $A\boldsymbol{y} = \boldsymbol{0}$ and $A\boldsymbol{y}' = \boldsymbol{0}$, we can show that $\boldsymbol{y}^T(A \oplus A^T)\boldsymbol{y}' = \boldsymbol{y}^T A\boldsymbol{y}' \oplus \boldsymbol{y}^T A^T\boldsymbol{y}' = \boldsymbol{y}^T A\boldsymbol{y}' \oplus (A\boldsymbol{y})^T\boldsymbol{y}' = 0$, which implies that $\mathcal{S}\boldsymbol{y}$ commutes with $\mathcal{S}\boldsymbol{y}'$.

□

Based on Proposition 4 we can show that the optimal number of internal Hadamard gates is equal to $\text{rank}(A)$. Let $\mathcal{S}'$ be a sequence of Pauli products such that the columns of $\mathcal{S}'$ are forming a spanning set of $\{\mathcal{S}\boldsymbol{y} \mid A\boldsymbol{y} = \boldsymbol{0}, \boldsymbol{y} \in \mathbb{F}_2^m\}$. It follows that for all $\boldsymbol{y}$ satisfying $A\boldsymbol{y} = \boldsymbol{0}$ there exists a vector $\boldsymbol{y}'$ such that $\mathcal{S}\boldsymbol{y} = \mathcal{S}'\boldsymbol{y}'$. Moreover, Proposition 4 entails that all the Pauli products of $\mathcal{S}'$ are mutually commuting. Therefore if the Pauli products encoded in $\mathcal{S}'$ were all to be diagonal, then, for all $\boldsymbol{y}$ satisfying $A\boldsymbol{y} = \boldsymbol{0}$, the Pauli product $\mathcal{S}\boldsymbol{y}$ would be diagonal, i.e. $\mathcal{X}\boldsymbol{y} = \boldsymbol{0}$. Let C' be the circuit resulting from the execution of Algorithm 5 when $\mathcal{S}'$ is given as input and let $\tilde{\mathcal{S}}$ be the sequence of Pauli products where, for all i, the Pauli product encoded by $\tilde{\mathcal{S}}_{:,i}$ is equal to the Pauli product encoded by $\mathcal{S}_{:,i}$ conjugated by the Clifford operator associated with C'. Let $\tilde{M} = \begin{bmatrix}\tilde{\mathcal{X}}\\ A\end{bmatrix}$, for all $\boldsymbol{y}$ satisfying $A\boldsymbol{y} = \boldsymbol{0}$ we have $\tilde{\mathcal{X}}\boldsymbol{y} = \boldsymbol{0}$ because C' performs a simultaneous diagonalization on the Pauli products of $\mathcal{S}'$, as stated by Proposition 3. Consequently we have $\tilde{M}\boldsymbol{y} = \boldsymbol{0}$ for all $\boldsymbol{y} \in \text{nullspace}(A)$ and so $\text{rank}(\tilde{M}) = \text{rank}(A)$. Then we can use Algorithm 5 to produce a Clifford circuit $\tilde{C}$ that is a diagonalization network for $\tilde{\mathcal{S}}$ and such that $h(\tilde{C}) = \text{rank}(\tilde{M}) = \text{rank}(A)$. It follows that the Clifford circuit $C' :: \tilde{C}$ is a diagonalization network for $\mathcal{S}$ containing $h(\tilde{C}) = \text{rank}(A)$ internal Hadamard gates.

To solve the Internal-H-Opt problem optimally it is then essential to find a spanning set of

$\{\mathcal{S}\boldsymbol{y} \mid A\boldsymbol{y} = \boldsymbol{0}, \boldsymbol{y} \in \mathbb{F}_2^m\}$, which we encode in the columns of $\mathcal{S}'$. Constructing such a spanning set naively by finding all $\boldsymbol{y} \in \mathbb{F}_2^m$ satisfying $A\boldsymbol{y} = \boldsymbol{0}$ would imply a complexity of $\mathcal{O}(m^3)$ using a Gaussian elimination procedure, which is more computationally expensive than minimizing the number of Hadamard gates via Algorithm 5 in the case where $n < m$. Fortunately, we can actually rely on Algorithm 5 to compute $\mathcal{S}'$ with a complexity of $\mathcal{O}(n^2 m)$, as it is done in Algorithm 6. Indeed, if Algorithm 5 is used to constructed a diagonalization network C for the sequence of Pauli products $\mathcal{S}J_m$, then the stabilizer generators of the Clifford operator implemented by C are forming a spanning set of $\{\mathcal{S}\boldsymbol{y} \mid A\boldsymbol{y} = \boldsymbol{0}, \boldsymbol{y} \in \mathbb{F}_2^m\}$. We demonstrate this statement via the following proposition.

Proposition 5. *Let $\mathcal{S} = \begin{bmatrix} \mathcal{Z} \\ \mathcal{X} \end{bmatrix}$ be a sequence of m Pauli products, J_m be an exchange matrix of size $m \times m$ and let U be the Clifford operator associated with the Clifford circuit C produced by Algorithm 5 when $\mathcal{S}J_m$ is given as input. Let $\tilde{\mathcal{S}}$ be the sequence of Pauli products obtained by conjugating all the Pauli products of $\mathcal{S}$ by U, then $\tilde{\mathcal{X}} J_m \boldsymbol{y} = \boldsymbol{0}$ for all $\boldsymbol{y}$ satisfying $\boldsymbol{y}^T A^{(\mathcal{S}J_m)} = \boldsymbol{0}$, where $A^{(\mathcal{S}J_m)}$ is the commutativity matrix associated with $\mathcal{S}J_m$.*

Proof. Let $C^{(i)}$ be the circuit obtained after the ith recursive call to Algorithm 5 when $\mathcal{S}J_m$ is given as input, as such $C^{(i)}$ is a diagonalization network for the first $i+1$ columns of SJ_m. And let $\mathcal{S}^{(i)} = \begin{bmatrix} \mathcal{Z}^{(i)} \\ \mathcal{X}^{(i)} \end{bmatrix}$ be the sequence of Pauli products resulting from conjugating $\mathcal{S}$ by the Clifford operator associated with the circuit $C^{(i)}$. We defined $\boldsymbol{y}^{(i)} \in \mathbb{F}_2^m$ as follows:

$$y_j^{(i)} = \begin{cases} y_j & \text{if } j \leq i, \\ 0 & \text{otherwise,} \end{cases} \tag{6.14}$$

where $\boldsymbol{y} \in \mathbb{F}_2^m$ satisfies $\boldsymbol{y}^T A^{(\mathcal{S}J_m)} = \boldsymbol{0}$.

In the case where $i = 0$, the equality $\mathcal{X}^{(0)} J_m \boldsymbol{y}^{(0)} = \boldsymbol{0}$ is satisfied because the Pauli product encoded by the first column of $\mathcal{S}^{(0)} J_m$ is diagonal and $y_j^{(0)} = 0$ for all $j > 0$. More generally, the Pauli product encoded by the ith column of $\mathcal{S}^{(i)} J_m$ is diagonal, and so the following holds:

$$\bigoplus_{k \in K} \mathcal{X}_k^{(i)} J_m = A_i^{(\mathcal{S}J_m)} \oplus A_{:,i}^{(\mathcal{S}J_m)} \tag{6.15}$$

where $K = \{k \mid \mathcal{Z}_{k,m-i-1}^{(i)} = 1\}$. Here the ith column of $A^{(\mathcal{S}J_m)}$ must be added to the ith row of $A^{(\mathcal{S}J_m)}$ to form the vector describing how the ith Pauli rotation commutes or anticommutes with the other Pauli rotations of the sequence. In this sense, it is a generalization of Equation 6.8 to the other rows of $A^{(\mathcal{S}J_m)}$. Equation 6.15 entails

$$\left[\bigoplus_{k \in K} \mathcal{X}_k^{(i)} J_m\right]^T \boldsymbol{y}^{(i)} = \left[A_i^{(\mathcal{S}J_m)} \oplus A_{:,i}^{(\mathcal{S}J_m)}\right]^T \boldsymbol{y}^{(i)} \tag{6.16}$$

Moreover, we have $\left[A_i^{(\mathcal{S}J_m)}\right]^T \boldsymbol{y}^{(i)} = 0$ because $A_{i,j}^{(\mathcal{S}J_m)} = 0$ for all $j \leq i$ and $y_j^{(i)} = 0$ for all $j > i$. And we also have $\left[A_{:,i}^{(\mathcal{S}J_m)}\right]^T \boldsymbol{y}^{(i)} = 0$ because $\left[A_{:,i}^{(\mathcal{S}J_m)}\right]^T \boldsymbol{y}^{(i)} = \left[A_{:,i}^{(\mathcal{S}J_m)}\right]^T \boldsymbol{y}$ as $A_{j,i}^{(\mathcal{S}J_m)} = 0$ for all $j > i$ and $\left[A_{:,i}^{(\mathcal{S}J_m)}\right]^T \boldsymbol{y} = \boldsymbol{y}^T A_{:,i}^{(\mathcal{S}J_m)} = 0$ by definition. Thus, we proved that the following holds:

$$\left[\bigoplus_{k \in K} \mathcal{X}_k^{(i)} J_m\right]^T \boldsymbol{y}^{(i)} = 0 \tag{6.17}$$

where $K = \{k \mid \mathcal{Z}^{(i)}_{k,m-i-1} = 1\}$.

Let's assume that $\mathcal{X}^{(i)} J_m \boldsymbol{y}^{(i)} = \mathbf{0}$, we can then distinguish two cases for the $(i+1)$th iteration of Algorithm 5. In the case where the $(i+1)$th Pauli product of $\mathcal{S}^{(i)} J_m$ is diagonal, the circuit $C^{(i+1)}$ can be obtained from $C^{(i)}$ by appending a $\{\text{CNOT}, S\}$ circuit to it. If the Pauli product encoded by $\mathcal{S}^{(i)} J_m \boldsymbol{y}^{(i)}$ is diagonal, as we assumed, then the Pauli product encoded by the vector $\mathcal{S}^{(i+1)} J_m \boldsymbol{y}^{(i)}$ is also diagonal as no Hadamard gate was appended to $C^{(i)}$ to derive $C^{(i+1)}$ from it. In addition, the $(i+1)$th Pauli product of $\mathcal{S}^{(i+1)} J_m$ is also diagonal which imply that the Pauli product encoded by the vector $\mathcal{X}^{(i+1)} J_m \boldsymbol{y}^{(i+1)}$ is diagonal and so $\mathcal{X}^{(i+1)} J_m \boldsymbol{y}^{(i+1)} = \mathbf{0}$. Therefore, in such case where the $(i+1)$th Pauli product of $\mathcal{S}^{(i)} J_m$ is diagonal, the equality $\mathcal{X}^{(i)} J_m \boldsymbol{y}^{(i)} = \mathbf{0}$ implies that $\mathcal{X}^{(i+1)} J_m \boldsymbol{y}^{(i+1)} = \mathbf{0}$.

In the case where the $(i+1)$th Pauli product of $\mathcal{S}^{(i)} J_m$ is not diagonal, the circuit $C^{(i+1)}$ can be constructed from $C^{(i)}$ by appending a $\{\text{CNOT}, S\}$ circuit to it and a final Hadamard gate on some qubit j. Let $\hat{C}^{(i+1)}$ be the circuit resulting from appending this $\{\text{CNOT}, S\}$ circuit to $C^{(i)}$, i.e. $\hat{C}^{(i+1)}$ corresponds to the circuit $C^{(i+1)}$ whose last gate, which is a Hadamard gate, has been removed. Let $\hat{\mathcal{S}}^{(i+1)}$ be the sequence of Pauli products obtained by conjugating all the Pauli products of $\mathcal{S}$ by the Clifford operator associated with $\hat{C}^{(i+1)}$. Using the same reasoning as before, if the Pauli product encoded by $\mathcal{S}^{(i)} J_m \boldsymbol{y}^{(i)}$ is diagonal then the Pauli product encoded by the vector $\hat{\mathcal{S}}^{(i+1)} J_m \boldsymbol{y}^{(i)}$ is also diagonal as no Hadamard gate was appended to $C^{(i)}$ to derive $\hat{C}^{(i+1)}$ from it, and so we have $\hat{\mathcal{X}}^{(i+1)} J_m \boldsymbol{y}^{(i)} = \mathbf{0}$.

The circuit $C^{(i+1)}$ can be obtained from $\hat{C}^{(i+1)}$ by appending a Hadamard gate to it on some qubit j. Therefore, $\mathcal{X}^{(i+1)}_k = \hat{\mathcal{X}}^{(i+1)}_k$ for all $k \neq j$, and so

$$\left[\mathcal{X}^{(i+1)}_k J_m\right]^T \boldsymbol{y}^{(i)} = \left[\hat{\mathcal{X}}^{(i+1)}_k J_m\right]^T \boldsymbol{y}^{(i)} = 0 \tag{6.18}$$

for all $k \neq j$. The $(i+1)$th Pauli product of $\mathcal{S}^{(i+1)} J_m$ is diagonal which means that the $(i+1)$th column of $\mathcal{X}^{(i+1)} J_m$ is equal to $\mathbf{0}$, and so the equality holds as well for $\boldsymbol{y}^{(i+1)}$:

$$\left[\mathcal{X}^{(i+1)}_k J_m\right]^T \boldsymbol{y}^{(i+1)} = \left[\mathcal{X}^{(i+1)}_k J_m\right]^T \boldsymbol{y}^{(i)} = 0 \tag{6.19}$$

for all $k \neq j$. Notice that $j \in K$ where $K = \{k \mid \mathcal{Z}^{(i+1)}_{k,m-i-1} = 1\}$, then from Equation 6.17 we can infer that

$$\left[\mathcal{X}^{(i+1)}_j J_m\right]^T \boldsymbol{y}^{(i+1)} \oplus \left[\bigoplus_{k \in \hat{K}} \mathcal{X}^{(i+1)}_k J_m\right]^T \boldsymbol{y}^{(i+1)} = 0 \tag{6.20}$$

where $\hat{K} = K \setminus \{j\}$. From Equation 6.19 we can deduce that the second term of Equation 6.20 is equal to 0, therefore we have

$$\left[\mathcal{X}^{(i+1)}_j J_m\right]^T \boldsymbol{y}^{(i+1)} = 0 \tag{6.21}$$

which, when combined with Equation 6.19, entails $\mathcal{X}^{(i+1)} J_m \boldsymbol{y}^{(i+1)} = \mathbf{0}$ and concludes the proof of Proposition 5.

□

We can now demonstrate Theorem 10 on the basis of Proposition 5.

Proof of Theorem 10. Let $\mathcal{S}'$ be as defined in Algorithm 6 and let C be the circuit produced by Algorithm 6 when $\mathcal{S} = \begin{bmatrix} \mathcal{Z} \\ \mathcal{X} \end{bmatrix}$ is given as input. As C is a diagonalization network for $\begin{bmatrix} \mathcal{S}' & \mathcal{S} \end{bmatrix}$ it

can be splitted in two subcircuits such that $C = C_1 :: C_2$, where C_1 and C_2 are diagonalization networks for $\mathcal{S}'$ and $\tilde{\mathcal{S}}$ respectively with $\tilde{\mathcal{S}} = \begin{bmatrix} \tilde{\mathcal{Z}} \\ \tilde{\mathcal{X}} \end{bmatrix} = U^\dagger \mathcal{S} U$ where U is the Clifford operator associated with C_1. The number of internal Hadamard gates in C is therefore equal to the number of Hadamard gates in C_2, proving Theorem 10 can then be done by proving that $h(C_2) = \text{rank}(\tilde{M}) = \text{rank}(A^{(\mathcal{S})})$ where $\tilde{M} = \begin{bmatrix} \tilde{\mathcal{X}} \\ A^{(\mathcal{S})} \end{bmatrix}$.

The Pauli products encoded in the matrix $\mathcal{S}J_m$ are the same as in $\mathcal{S}$ but in reverse order. Consequently we have $A^{(\mathcal{S})}_{i,j} = A^{(\mathcal{S}J_m)}_{m-j-1,m-i-1}$, therefore by reversing the order of the rows and columns of $A^{(\mathcal{S}J_m)}$ and transposing it to obtain a strictly upper triangular matrix we get the matrix $A^{(\mathcal{S})}$:

$$\left[J_m A^{(\mathcal{S}J_m)} J_m\right]^T = A^{(\mathcal{S})} \tag{6.22}$$

From this we can deduce that

$$\begin{aligned} & A^{(\mathcal{S})}\boldsymbol{y} = \mathbf{0} \\ \Rightarrow \quad & \left[J_m A^{(\mathcal{S}J_m)} J_m\right]^T \boldsymbol{y} = \mathbf{0} \\ \Rightarrow \quad & \boldsymbol{y}^T J_m A^{(\mathcal{S}J_m)} J_m = \mathbf{0} \\ \Rightarrow \quad & \overline{\boldsymbol{y}}^T A^{(\mathcal{S}J_m)} = \mathbf{0} \end{aligned} \tag{6.23}$$

where $\overline{\boldsymbol{y}} = J_m \boldsymbol{y}$. And based on Proposition 5 we have

$$\begin{aligned} & \tilde{\mathcal{X}} J_m \overline{\boldsymbol{y}} = \mathbf{0} \\ \Rightarrow \quad & \tilde{\mathcal{X}}\boldsymbol{y} = \mathbf{0} \end{aligned} \tag{6.24}$$

Thus, for all $\boldsymbol{y} \in \text{nullspace}(A^{(\mathcal{S})})$ we have $\tilde{\mathcal{X}}\boldsymbol{y} = \mathbf{0}$ and therefore $\tilde{M}\boldsymbol{y} = \mathbf{0}$, which implies that $h(C_2) = \text{rank}(\tilde{M}) = \text{rank}(A^{(\mathcal{S})})$ and concludes the proof of Theorem 10.

□

6.4 Improving the complexity

Algorithm 5 and 6 are taking a sequence of Pauli products $\mathcal{S}$ as input and output a diagonalization network for $\mathcal{S}$. In order to use these algorithms to minimize the number of Hadamard gates, or internal Hadamard gates, in a circuit C it is then required to first extract from C the sequence of Pauli products $\mathcal{S}$ for which the diagonalization network must be constructed. This procedure can be done with a complexity $\mathcal{O}(nM)$ by using a tableau, where n is the number of qubits and M is the number of gates in C. In this section, we will see how we can merge the extraction of the sequence of Pauli products $\mathcal{S}$ with our algorithms to obtain the desired re-synthesis of C with a complexity of $\mathcal{O}(nM + n^2 h)$ instead of $\mathcal{O}(nM + n^2 m)$ where m is the number of Pauli products in $\mathcal{S}$ and $h \leq m$ is the minimal number of Hadamard gates required to construct a diagonalization network for $\mathcal{S}$. We first explain our notations related to the tableau representation, commonly used to represent a Clifford operator. In Subsection 6.4.1 we present an algorithm which performs the re-synthesis of a sequence of Pauli rotations implemented by a given circuit up to a final Clifford operator and with a minimal number of Hadamard gates. Finally, in Subsection 6.4.2, we present an algorithm which produces a circuit that is a re-synthesis of a given circuit and which implements the same sequence of Pauli rotations but with a minimal

Algorithm 7: H-Opt

Input: A Clifford+R_Z circuit C and a tableau $\mathcal{T} = \begin{bmatrix} \boldsymbol{s}^T \\ \mathcal{Z} \\ \mathcal{X} \end{bmatrix}$.

Output: A circuit C_{out} and a tableau $\mathcal{T}_{out}$, such that C_{out} is a re-synthesis of C and implements the same sequence of Pauli rotations as C up to an initial and final Clifford operator represented by $\mathcal{T}$ and $\mathcal{T}_{out}^{-1}$ respectively.

procedure `HOpt`(C, $\mathcal{T}$)
  $C_{out} \leftarrow$ new empty circuit
  foreach gate $G \in C$ **do**
    if G is Clifford **then**
      Prepend $G^\dagger$ to $\mathcal{T}$
    end
    if G is a non-Clifford $R_{Z_k}(\theta)$ gate **then**
      if $\exists i$ such that $\mathcal{X}_{i,k} = 1$ **then**
        foreach $j \in \{j \mid \mathcal{X}_{j,k} = 1\} \setminus \{i\}$ **do**
          $C_{out} \leftarrow C_{out} :: \mathrm{CNOT}_{i,j}$
          Append $\mathrm{CNOT}_{i,j}$ to $\mathcal{T}$
        end
        if $\mathcal{Z}_{i,k} = 1$ **then**
          $C_{out} \leftarrow C_{out} :: S_i$
          Append S_i to $\mathcal{T}$
        end
        $C_{out} \leftarrow C_{out} :: H_i$
        Append H_i to $\mathcal{T}$
      end
      $i \leftarrow$ any value satisfying $\mathcal{Z}_{i,k} = 1$
      $\tilde{C} \leftarrow$ new empty circuit
      foreach $j \in \{j \mid \mathcal{Z}_{j,k} = 1\} \setminus \{i\}$ **do**
        $\tilde{C} \leftarrow \tilde{C} :: \mathrm{CNOT}_{j,i}$
      end
      if $s_k = 1$ **then**
        $\tilde{C} \leftarrow \tilde{C} :: X_i$
      end
      $C_{out} \leftarrow C_{out} :: \tilde{C} :: R_{Z_i}(\theta) :: \tilde{C}^{-1}$
    end
  end
  return $(C_{out}, \mathcal{T})$

Algorithm 8: Internal-H-Opt

Input: A Cliffod+R_Z circuit C.

Output: A circuit that is a re-synthesis of C and which implements the same sequence of Pauli rotations as C with a minimal number of Hadamard gates and internal Hadamard gates.

```
1 procedure InternalHOpt(C)
2     T ← new identity tableau
3     foreach Clifford gate G ∈ C do
4         Prepend G† to T
5     end
6     (C̃, T̃) ← HOpt(C^{-1}, T)
7     C_T̃ ← CliffordSynthesis(T̃)
8     (C_out, T_f) ← HOpt(C, T̃)
9     return C_T̃ :: C_out :: CliffordSynthesis(T_f^{-1})
```

number of Hadamard gates and internal Hadamard gates.

The tableau representation. As explained in Section 1.2.1.2, a tableau encodes $2n$ generators which can be represented by $2n$ independent Pauli products along with a phase for each one of these Pauli products. We can thus reuse our method of encoding for a sequence of Pauli products $\mathcal{S}$ and represent a tableau by a block matrix $\mathcal{T} = \begin{bmatrix} \boldsymbol{s}^T \\ \mathcal{Z} \\ \mathcal{X} \end{bmatrix}$ of size $(2n+1) \times 2n$ where n is the number of qubits. The first row of $\mathcal{T}$ corresponds to a vector $\boldsymbol{s} \in \{0,1\}^{2n}$ which encodes the phases of the generators, the subsequent n rows of $\mathcal{T}$ are forming the submatrix $\mathcal{Z}$ and the last n rows of $\mathcal{T}$ are forming the submatrix $\mathcal{X}$. The jth column of $\mathcal{T}$ is then encoding the jth generator: s_j encodes its phase which corresponds to $(-1)^{s_j}$ and $(\mathcal{Z}_{i,j}, \mathcal{X}_{i,j})$ encodes its ith Pauli matrix, such that the values $(0,0), (0,1), (1,1)$ and $(1,0)$ are corresponding to the Pauli matrices I, X, Y and Z respectively. The first n columns of $\mathcal{T}$ are encoding the stabilizer generators, whereas the last n columns of $\mathcal{T}$ are encoding the destabilizer generators. The identity tableau $\mathcal{T}$ associated with an empty circuit is such that the matrix $\begin{bmatrix} \mathcal{Z} \\ \mathcal{X} \end{bmatrix}$ is forming the identity matrix and $\boldsymbol{s} = \boldsymbol{0}$, said otherwise the ith stabilizer generator of $\mathcal{T}$ is Z_i and the ith destabilizer generator of $\mathcal{T}$ is X_i. The inverse tableau of $\mathcal{T}$, denoted by $\mathcal{T}^{-1}$, is the tableau associated with the Clifford operator $U^\dagger$ where U is the Clifford operator associated with $\mathcal{T}$. Analogously, the inverse of a circuit C, denoted C^{-1}, is the circuit obtained from C by replacing every gate G by $G^\dagger$ and by reversing the order of its gates. Let $\mathcal{S}$ be a sequence of Pauli products, if $\tilde{\mathcal{S}} = U^\dagger \mathcal{S} U$ then we will equivalently say that $\tilde{\mathcal{S}} = \mathcal{T}^{-1}\mathcal{S}\mathcal{T}$ where $\mathcal{T}$ is the tableau associated with the Clifford operator U.

Let C be a Clifford circuit such that its associated Clifford operator is represented by a tableau $\mathcal{T}$. If a Clifford gate from the set $\{\text{CNOT}, S, H\}$ is appended to C then the generators of $\mathcal{T}$ can be updated accordingly with $\mathcal{O}(n)$ operations, where n is the number of qubits. The operations to perform on the Pauli products encoded by $\mathcal{T}$ are the same as the one depicted in Figure 6.1, similar operations can be performed to update the phases associated with the Pauli products in $\mathcal{O}(n)$ [17]. Also, if a Clifford gate from the set $\{\text{CNOT}, S, H\}$ is prepended to C, then $\mathcal{T}$ can also be updated with $\mathcal{O}(n)$ operations [125]. When $\mathcal{T}$ is updated in such manner we will say that we append, or prepend, a gate to $\mathcal{T}$. As explained in Section 6.2.3, a Clifford

operator, represented by a tableau $\mathcal{T}$ and acting on n qubits, can be implemented over the $\{X, \text{CNOT}, S, H\}$ gate set with a complexity of $\mathcal{O}(n^3)$ and with a minimal number of Hadamard gates by first diagonalizing its stabilizer generators using Algorithm 5, and then finishing its synthesis using only $\{X, \text{CNOT}, S\}$ gates. We use the term `CliffordSynthesis` to denote this procedure in our algorithms.

6.4.1 H-Opt algorithm

Consider the algorithm whose pseudo-code is given in Algorithm 7 and which takes a circuit C and a tableau $\mathcal{T}_{in}$ as input, and let $\mathcal{S}$ be the sequence of Pauli products associated with the sequence of Pauli rotations implemented by C. This algorithm outputs a circuit C_{out} and a tableau $\mathcal{T}$ such that C_{out} is a re-synthesis of C and implements the same sequence of Pauli rotations as C up to an initial and final Clifford operator represented by $\mathcal{T}_{in}$ and $\mathcal{T}_{out}^{-1}$ respectively.

Algorithm 7 is composed of a loop iterating over the gates of C and which contains two distinct cases: either the current gate G is a Clifford gate or it is not. If G is Clifford gate then $G^\dagger$ is prepended into $\mathcal{T}$. If G is a non-Clifford $R_{Z_i}(\theta)$ gate then we must compute the Pauli rotation that should be appended to C_{out}. To do so we can first compute which Pauli rotation is actually being implemented by C by pulling all the Clifford gates preceding G through the Pauli rotation $R_{Z_i}(\theta)$. The Pauli rotation obtained is then $UR_{Z_i}(\theta)U^\dagger$ where U is the Clifford operator associated with the Clifford circuit composed of all the Clifford gates preceding G. Then, to be appended into C_{out}, the Pauli rotation must also be propagated through the initial tableau $\mathcal{T}_{in}$, we will denote V the Clifford operator associated with $\mathcal{T}_{in}$. Finally, the Pauli rotation must be propagated through all the Clifford gates that are in C_{out} so far, we denote W the associated Clifford operator. The Pauli rotation to append to the circuit C_{out} is therefore $W^\dagger V^\dagger U R_{Z_i}(\theta) U^\dagger V W$. We can notice that the Clifford operator $U^\dagger V W$ is in fact associated with the tableau $\mathcal{T}$. Indeed, $\mathcal{T}$ is initially equal to $\mathcal{T}_{in}$, the inverse Clifford gates that are preceding G in C has been prepended to $\mathcal{T}$ and the Clifford gates that are in C_{out} so far has been appended to $\mathcal{T}$. The Pauli operator P satisfying $R_P(\theta) = W^\dagger V^\dagger U R_{Z_i}(\theta) U^\dagger V W$ is therefore the ith stabilizer generator of $\mathcal{T}$, which is encoded by the ith column of $\mathcal{T}$.

The Pauli rotation $R_P(\theta)$ can then be implemented by first performing the synthesis of a Clifford operator U such that $U^\dagger R_P(\theta) U$ is diagonal, and then by performing the synthesis of a Clifford operator V satisfying $V^\dagger U^\dagger R_P(\theta) U V = R_{Z_i}(\theta)$, for some qubit i. The Clifford operator V can be synthesized using only $\{X, \text{CNOT}\}$ gates as $U^\dagger R_P(\theta) U$ is diagonal, this is done in Algorithm 7 by constructing the circuit $\tilde{C}$. Once the operators U and V have been implemented, the gate $R_{Z_i}(\theta)$ can be appended to the circuit. The operator $V^\dagger$ does not necessarily need to be implemented, but it is actually implemented in Algorithm 7 to avoid additional operations that would be required to update the tableau $\mathcal{T}$. We should not treat the operator $U^\dagger$ the same way as it would increase the number of Hadamard gates in the circuit, $U^\dagger$ is therefore not implemented in Algorithm 7 and the tableau $\mathcal{T}$ is updated accordingly by appending the gates realizing the implementation of U to it. Note that the method utilized to implement U is the same as the one in Algorithm 5, which uses exactly one Hadamard gate when P is not diagonal. It follows from the results in Section 6.2 that Algorithm 7 can be used to solve the H-Opt problem for $\tilde{\mathcal{S}} = \mathcal{T}_{in}^{-1} \mathcal{S} \mathcal{T}_{in}$. More concretely, by removing all the non-Clifford R_Z gates from the circuit produced by Algorithm 7 we obtain a diagonalization network which solves the H-Opt problem for $\tilde{\mathcal{S}}$.

Let C' and C'_{out} be the Clifford circuits obtained by removing all the non-Clifford R_Z gates from C and C_{out} respectively, and let $C_{\mathcal{T}_{in}}$ be a Clifford circuit whose Clifford operator is associated with the tableau $\mathcal{T}_{in}$. In the end of Algorithm 7, the tableau $\mathcal{T}$ is associated with the

Clifford operator implemented by the circuit $C_{\mathcal{T}} = C'^{-1} :: C_{\mathcal{T}_{in}} :: C'_{out}$. As C_{out} implements the sequence of Pauli rotations associated with $\tilde{\mathcal{S}} = \mathcal{T}_{in}^{-1}\mathcal{S}\mathcal{T}_{in}$ up to a final Clifford operator implemented by C'^{-1}_{out}, it follows that $C_f = C_{\mathcal{T}_{in}} :: C_{out} :: C_{\mathcal{T}}^{-1}$ is a re-synthesis of C and implements the same sequence of Pauli rotations as C. If the input tableau $\mathcal{T}_{in}$ is the identity tableau, or can be implemented with no Hadamard gates, and if $C_{\mathcal{T}}$ is implemented with a minimal number of Hadamard gates using the procedure described in Section 6.2.3, then C_f is a re-synthesis of C which implements the same sequence of Pauli rotations with a minimal number of Hadamard gates.

Complexity analysis. The main loop of Algorithm 7 is performing M iterations where M is the number of gates in the input circuit. At each iteration, if the current gate is a Clifford gate then it is prepended to $\mathcal{T}$ which is done in $\mathcal{O}(n)$ operations, where n is the number of qubits in the input circuit. If the current gate is a non-Clifford $R_{Z_k}(\theta)$ gate then the algorithm append $\mathcal{O}(n)$ gates to C_{out}. In the case where the kth stabilizer generator of $\mathcal{T}$ is not diagonal then a subset of these gates are appended to $\mathcal{T}$ which takes $\mathcal{O}(n)$ operations for each gates. This happens exactly h times where h is the number of Hadamard gates in the output circuit C_{out}, which implies a cost of $\mathcal{O}(n^2h)$ operations. Thus, the overall complexity of Algorithm 7 is $\mathcal{O}(nM + n^2h)$.

6.4.2 Internal-H-Opt algorithm

Algorithm 8 is based on the procedure explained in Section 6.3 and utilized by Algorithm 6 to synthesize a diagonalization network for a sequence of Pauli products with a minimal number of internal Hadamard gates. It takes a Clifford+R_Z circuit C as input and outputs a circuit which is a re-synthesis of C and which implements the same sequence of Pauli rotations as C with a minimal number of Hadamard gates and internal Hadamard gates.

As explained in Section 6.3, in order to solve the Internal-H-Opt problem for a sequence of m Pauli products $\mathcal{S}$ it is necessary to find a Clifford operator U that minimizes $\text{rank}(\tilde{M})$ where $\tilde{M} = \begin{bmatrix} \tilde{\mathcal{X}} \\ A^{(\mathcal{S})} \end{bmatrix}$, $A^{(\mathcal{S})}$ is the commutativity matrix associated with $\mathcal{S}$ and $\tilde{\mathcal{S}} = \begin{bmatrix} \tilde{\mathcal{Z}} \\ \tilde{\mathcal{X}} \end{bmatrix} = U^\dagger \mathcal{S} U$. We proved that the Clifford operator associated with the circuit produced by Algorithm 5 when $\mathcal{S}J_m$, where J_m is an exchange matrix of size $m \times m$, is given as input is satisfying this property. Let $\mathcal{S}$ be a sequence of m Pauli products associated with the sequence of Pauli rotations implemented by a Clifford+R_Z circuit C, then the Clifford operator U described above can be computed by the `HOpt` procedure described in Algorithm 7. To do so, the circuit C^{-1} and the tableau $\mathcal{T}$ must be given as input to the `HOpt` procedure, such that $\mathcal{T}$ is the tableau associated with the Clifford operator implemented by the circuit C'^{-1} where C' is the Clifford circuit obtained by removing all the non-Clifford R_Z gates of C. The circuit C^{-1} is provided so that the Pauli rotations are processed in reversed order by the `HOpt` procedure. For the tableau $\mathcal{T}$, it must be provided because the circuit C^{-1} does not necessarily implements the same sequence of Pauli rotations as C, however the circuit $C' :: C^{-1}$ do implement the same sequence of Pauli rotations as the circuit C. We can be convinced by this fact by noticing that the Clifford operator formed by all the Clifford gates preceding a non-Clifford gate in C is the same as the Clifford operator formed by all the Clifford gates preceding the corresponding non-Clifford gate in $C' :: C^{-1}$. Then, as shown in Section 6.4.1, when the `HOpt` procedure is executed with C^{-1} and $\mathcal{T}$ as parameters, it will produce a circuit $\tilde{C}$ and a tableau $\tilde{\mathcal{T}}$ associated with the Clifford operator implemented by the circuit $C' :: C'^{-1} :: \tilde{C}'$, which is equivalent to the circuit $\tilde{C}'$, and where $\tilde{C}'$ is the Clifford circuit obtained by removing all the non-Clifford R_Z gates from $\tilde{C}$. The circuit $\tilde{C}'$ then solves the H-Opt problem for $\mathcal{S}J_m$, and is an implementation of the Clifford operator associated with

the tableau $\tilde{\mathcal{T}}$. From the results of Section 6.3, it follows that if $\tilde{\mathcal{S}} = \begin{bmatrix} \tilde{\mathcal{Z}} \\ \tilde{\mathcal{X}} \end{bmatrix} = \tilde{\mathcal{T}}^{-1}\mathcal{S}\tilde{\mathcal{T}}$ then $\text{rank}(\tilde{M}) = \text{rank}(A^{(\mathcal{S})})$ where $\tilde{M} = \begin{bmatrix} \tilde{\mathcal{X}} \\ A^{(\mathcal{S})} \end{bmatrix}$.

Algorithm 8 then performs the synthesis of the Clifford operator associated with $\tilde{\mathcal{T}}$ with a minimal number of Hadamard gates, the Clifford circuit $C_{\tilde{\mathcal{T}}}$ obtained will be the initial Clifford circuit of the circuit produced by Algorithm 8. The algorithm then calls a second time the `HOpt` procedure with C and $\tilde{\mathcal{T}}$ given as parameters in order to implement the sequence of Pauli rotations associated with $\tilde{\mathcal{S}}$ with a minimal number of internal Hadamard gates and up to a final Clifford circuit. The tableau $\tilde{\mathcal{T}}$ must be given as input so that the sequence of Pauli rotations implemented is the one associated with the sequence of Pauli products $\tilde{\mathcal{S}}$ and not $\mathcal{S}$. The procedure `HOpt` will then produce a circuit C_{out} and a tableau $\mathcal{T}_f$ such that C'_{out} is solving the H-Opt problem for $\tilde{\mathcal{S}}$ and $\mathcal{T}_f$ is associated with the Clifford operator implemented by $C_f = C'^{-1} :: C_{\tilde{\mathcal{T}}} :: C'_{out}$, where C' and C'_{out} are the circuits obtained by removing all the non-Clifford R_Z gates from C and C_{out} respectively. We can then deduce that $C_{\tilde{\mathcal{T}}} :: C_{out} :: C_f^{-1}$ is implementing the same sequence of Pauli rotations as C and the Clifford operator formed by all the Clifford gates of this circuit is the same as the Clifford operator formed by all the Clifford gates of C. Thus, the circuit produced by Algorithm 8 is a re-synthesis of C and it implements the same sequence of Pauli rotations with a minimal number of Hadamard gates and internal Hadamard gates.

Complexity analysis. Let h be the number of Hadamard gates within the circuit produced by Algorithm 8, and let n be the number of qubits in C. The algorithm performs two calls to the `HOpt` procedure for C^{-1} and C respectively, which both contains M gates. The first call, with C^{-1} given as input, will produce a circuit which contains $\tilde{h}$ number of Hadamard gates, such that $\tilde{h} \leq h$. The second call, with C given as input, will produce a circuit which contains a number of Hadamard gates that is equal to the number of internal Hadamard gates in the circuit produced by Algorithm 8, and which is therefore less than or equal to h. Hence, these two calls to Algorithm 7 have a cost of $\mathcal{O}(nM + n^2h)$ operations. The procedure `CliffordSynthesis` is also called two times, which induces a cost of $\mathcal{O}(n^3)$ operations. Thus, the overall complexity of Algorithm 8 is $\mathcal{O}(nM + n^2h + n^3)$, which corresponds to $\mathcal{O}(nM + n^2h)$ in the typical case where $h > n$.

Note that the two calls to the `CliffordSynthesis` procedure can be avoided if the objective is to minimize the number of internal Hadamard gates in the circuit and not the number of Hadamard gates. Indeed, the first call to the `HOpt` procedure will produce a circuit $\tilde{C}$ and a tableau $\mathcal{T}$ such that $\mathcal{T}$ is associated with the Clifford operator implemented by $\tilde{C}'$ where $\tilde{C}'$ is obtained by removing all the non-Clifford R_Z gates from $\tilde{C}$. Performing the synthesis of $\mathcal{T}$ will therefore produce a circuit that is equivalent to $\tilde{C}'$. Consequently, instead of calling the procedure `CliffordSynthesis`, an equivalent circuit could be obtained by constructing $\tilde{C}'$ which can be done with $\mathcal{O}(nM)$ operations as $\tilde{C}$ contains $\mathcal{O}(nM)$ gates. Of course, the drawbacks of this method are that $\tilde{C}'$ may not contain an optimal number of Hadamard gates and that the worst-case complexity would be greater than $\mathcal{O}(n^3)$ in the case where $M > n^2$. The second call to `CliffordSynthesis` can also be avoided in a similar manner. Indeed, $\mathcal{T}_f$ is associated with the Clifford operator implemented by $C_f = C'^{-1} :: C_{\tilde{\mathcal{T}}} :: C'_{out}$, where C' and C'_{out} are the circuits obtained by removing all the non-Clifford R_Z gates from C and C_{out} respectively. The circuit C_f can then be constructed in $\mathcal{O}(nM)$ as the circuits C'^{-1}, $C_{\tilde{\mathcal{T}}}$ and C'_{out} all contain $\mathcal{O}(nM)$ gates. Thus, we can design an algorithm which produces a circuit $\hat{C}$ with a complexity of $\mathcal{O}(nM + n^2h)$,

even in the case where $h < n$, and such that $\hat{C}$ is a re-synthesis of a Clifford+R_Z circuit C and implements the same sequence of Pauli rotations as C but with a minimal number of internal Hadamard gates.

6.5 Benchmarks

We compare the performances of Algorithm 8, the `InternalHOpt` procedure, to the `moveH` procedure presented in Reference [112] and which has a complexity of $\mathcal{O}(M^2)$ where M is the number of gates in the input circuit. Note that the `moveH` procedure does not include the T gates reduction method of Reference [112] based on spider nest identities and which is normally performed once the number of Hadamard gates have been reduced. The `moveH` procedure applies a sequence of rewriting rules on the circuit with the aim of reducing the number of internal Hadamard gates. During this process the number of T gates may also be reduced, which modifies the sequence of Pauli rotations implemented by the circuit. This can then lead to a better reduction in the number of internal Hadamard gates than the one obtained when only the `InternalHOpt` procedure is performed. Which is why, in order to better exploit the `InternalHOpt` procedure, it can be helpful to first execute an algorithm which can reduce the number of T gates in the circuit quickly and efficiently. The T-count reduction algorithms that are closest to these requirements are the provided in Reference [110] and in Reference [109], these two algorithms have in fact been proven to be equivalent [126]. The method used in these algorithms consists in merging the Pauli rotations in the sequence that are equivalent and that are not separated by another Pauli rotation with which they anticommute. We implemented the algorithm provided in Reference [110] such that it is not increasing the number of gates in the circuit in order to not increase the execution time of the `InternalHOpt` procedure. This procedure, which we refer to as `TMerge`, has a complexity of $\mathcal{O}(nM + nm^2)$ where n is the number of qubits, M is the number of gates in the input circuit and m is the number of Pauli rotations. If the T gates reduction rules used in the `moveH` subroutine is only consisting in merging two adjacent R_Z gates together, then we can infer that the number of T gates in the circuit after `moveH` procedure has been performed is always higher or equal to the number of T gates in the circuit after the `TMerge` procedure has been performed; this is corroborated by the results of our benchmarks.

We evaluate the different methods on a set of commonly used circuits which were obtained from Reference [127] and Reference [128]. We extended the set of circuits over which the benchmarks are performed by adding larger quantum circuit to better test the scalability of the different approach on various types of circuits. We added large adders circuits which are performing an addition over two registers of size 1024, 2048 and 4096 qubits, the implementation of these circuits is based on Reference [29]. We also added a circuit, given in Reference [129], that is an implementation of the block cipher DEFAULT. Finally, we added quantum circuits implementing the modular exponentiation part of Shor's algorithm for number factoring over 4, 8, 16 and 32 bits.

The `TMerge` and `InternalHOpt` procedures were implemented with the Rust programming language, while the `moveH` procedure was extracted from the implementation realized in Haskell by the authors of the method [130]. Our implementation of the `InternalHOpt` procedure used for the benchmarks is publicly available [8], along with the circuits used in the benchmarks and which have a reasonable size. The operations performed by the `InternalHOpt` algorithm mostly consist in bitwise operations between vectors in order to update the tableau. Thus, our algorithm can greatly benefits from SIMD (Same Instruction Multiple Data) instructions which enable the simultaneous execution of some of these bitwise operations. This have for example been used in

	`InternalHOpt`			`TMerge` [110] + `InternalHOpt`			`moveH` [112]		
Circuit	H-count	T-count	Time (s)	H-count	T-count	Time (s)	H-count	T-count	Time (s)
Tof_3	2	21	0.00	2	15	0.00	2	15	0.00
Tof_4	4	35	0.00	4	23	0.00	4	23	0.00
Tof_5	6	49	0.00	6	31	0.00	6	31	0.00
Tof_{10}	16	119	0.00	16	71	0.00	16	71	0.00
Barenco Tof_3	3	28	0.00	3	16	0.00	3	16	0.00
Barenco Tof_4	7	56	0.00	7	28	0.00	7	28	0.00
Barenco Tof_5	11	84	0.00	11	40	0.00	11	40	0.00
Barenco Tof_{10}	31	224	0.00	31	100	0.01	31	100	0.00
Mod5_4	0	28	0.00	0	8	0.00	0	8	0.00
VBE Adder_3	4	70	0.00	4	24	0.00	4	24	0.00
CSLA MUX_3	6	70	0.00	6	62	0.00	6	62	0.00
CSUM MUX_9	12	196	0.00	12	84	0.01	12	84	0.00
QCLA Com_7	18	203	0.00	18	95	0.01	18	95	0.00
QCLA Mod_7	58	413	0.00	58	237	0.02	58	237	0.00
QCLA Adder_{10}	25	238	0.00	25	162	0.01	25	162	0.00
Adder_8	41	399	0.00	37	173	0.02	41	215	0.01
Mod Adder_{1024}	304	1995	0.00	304	1011	0.12	304	1011	0.06
RC Adder_6	10	77	0.00	10	47	0.00	10	47	0.00
Mod Red_{21}	17	119	0.00	17	73	0.00	17	73	0.00
Mod Mult_{55}	3	49	0.00	3	35	0.00	3	35	0.00
GF(2^4) Mult	0	112	0.00	0	68	0.00	0	68	0.00
GF(2^5) Mult	0	175	0.00	0	115	0.01	0	115	0.00
GF(2^6) Mult	0	252	0.00	0	150	0.01	0	150	0.00
GF(2^7) Mult	0	343	0.00	0	217	0.02	0	217	0.01
GF(2^8) Mult	0	448	0.00	0	264	0.04	0	264	0.02
GF(2^9) Mult	0	567	0.00	0	351	0.05	0	351	0.03
GF(2^{10}) Mult	0	700	0.00	0	410	0.07	0	410	0.04
GF(2^{16}) Mult	0	1792	0.01	0	1040	0.43	0	1040	0.14
GF(2^{32}) Mult	0	7168	0.05	0	4128	7.19	0	4128	0.98
GF(2^{64}) Mult	0	28672	0.19	0	16448	125.07	0	16448	7.46
GF(2^{128}) Mult	0	114688	1.20	0	65664	2294.64	0	65664	60.47
GF(2^{256}) Mult	0	458752	8.22	0	262400	41474.34	0	262400	2922.20
GF(2^{512}) Mult	0	1835008	53.85	-	-	-	0	1049088	59186.15
Adder_{1024}	2044	14322	3.57	2044	8184	31.08	2046	8184	6.12
Adder_{2048}	4092	28658	18.98	4092	16376	179.07	4094	16376	25.69
Adder_{4096}	8188	57330	90.46	8188	32760	1182.67	8190	32760	131.11
DEFAULT	11936	62720	13.72	11936	39744	39.33	12030	39744	1602.60
Shor_4	9780	68320	0.21	5010	17052	5.91	9829	22514	77.52
Shor_8	69759	489741	1.74	35585	121341	158.91	69759	163827	6895.79
Shor_{16}	537630	3755115	15.80	312274	1042881	2821.94	-	-	-
Shor_{32}	4173389	29622691	172.98	387103	1303156	24150.54	-	-	-

Table 6.1: Comparison of different methods for the optimization of the number of internal Hadamard gates. The H-count corresponds to the number of internal Hadamard gates. A blank entry indicates that the execution couldn't be carried out in less than a day.

the CHP stabilizer circuit simulator [17]. We also exploit this concept in our implementation of the `InternalHOpt` procedure by using 256 bit wide Advanced Vector Extensions (AVX).

6.5.1 Benchmarks analysis

The results of our benchmarks are presented in Table 6.1. We can notice that the `InternalHOpt` procedure outperforms the `moveH` procedure in term of execution time on some circuits of large size. For instance, the Shor_{32} circuit was optimized in 173 seconds by the `InternalHOpt` procedure while the two other methods did not succeed in optimizing the circuit in less than a day. However, the `InternalHOpt` procedure alone does not always achieve the best results in the number of internal Hadamard gates. For the set of circuits and methods considered, the method achieving the best results in term of internal Hadamard gates is the `TMerge` + `InternalHOpt` approach. Indeed, the `TMerge` + `InternalHOpt` approach always leads to a number of internal Hadamard gates that is lower or equal to the numbers obtained by the `moveH` procedure. This fact also holds for the number of T gates. However, for some circuits, the performances of the `moveH` procedure and the `TMerge` + `InternalHOpt` approach are similar with respect to the H-count and T-count metrics, but the execution time of the `moveH` procedure is much lower. This is notably the case for the adder circuits of large size. These adder circuits have a low depth and a high number of qubits, which is far from the ideal case for `TMerge` + `InternalHOpt` approach since the complexity of both procedures is dependent on the number of qubits. On the contrary, the `moveH` procedure is not affected by the number of qubits as it has a complexity of $\mathcal{O}(M^2)$ where M is the number of gates within the circuit. This explains why the `moveH` procedure is competitive for these adder circuits and has an execution time that is close to the one of the `InternalHOpt` procedure.

Another series of circuits for which the `moveH` procedure is much faster than the `TMerge` + `InternalHOpt` approach are the "GF(2^n) Mult" circuits. This behaviour can be explained by analyzing the structure of the "GF(2^n) Mult" circuits and the design of the `TMerge` algorithm. The "GF(2^n) Mult" circuits are all implementing a sequence of Pauli rotations that are mutually commuting, which is why no internal Hadamard gate is required for these circuits. In the worst case, for every pair of Pauli rotations, the `TMerge` procedure will check whether two Pauli rotations commute or not. This routine, which seems unnecessary in the case where we know that the Pauli rotations are all mutually commuting, is particularly expensive for the "GF(2^n) Mult" circuits for which n is high since the number of Pauli rotations increases drastically with respect to n.

6.5.2 Outlook

Our primary motivation for optimizing the number of internal Hadamard gate is to foster the minimization of T-gates. Conversely, our benchmarks show that optimizing the number of T-gate leads to better minimization in the number of internal Hadamard gates. This interdependence between the T-count and H-count minimization problems could lead us to think that a second round of T-count optimization followed by a H-count optimization could lead to a lower number of internal Hadamard gates. Our investigations on that second round of optimization have not be fruitful as we did not succeed in reducing the number of internal Hadamard gates below the numbers obtained by the `TMerge` + `InternalHOpt` approach. It seems that once the `TMerge` procedure has been performed, it becomes difficult to modify the underlying sequence of Pauli rotations in such a way that it enables further reduction in the number of internal Hadamard gates. Our conclusion here is only based on some of our tests, more investigations with a wide

variety of T-count optimizers should be performed to know whether or not this second round of optimization could lead to an improvement in the number of internal Hadamard gates.

Two lines of investigations on how to perform the optimization of internal Hadamard gates more efficiently can be drawn out from these benchmarks. Firstly, the `TMerge` procedure is outperformed, with respect to the execution time, by the `moveH` procedure on some circuits such as the "GF(2^n) Mult" circuits, can the complexity of the `TMerge` procedure be improved so that it is more competitive on these circuits? Secondly, is it possible to design an algorithm similar to the `moveH` procedure, so that it has approximatively the same execution time, but which systematically obtains the same number of T gates as the `TMerge` procedure and which optimally minimizes the number of internal Hadamard gates in the resulting sequence of Pauli rotations as done by the `InternalHOpt` procedure?

Chapter 7

Angle-Agnostic Optimization of Non-Clifford Rotation Gates

Non-Clifford rotation gates are typically more resource-intensive to implement fault-tolerantly than Clifford gates. Finding efficient strategies to optimize the number of these gates is therefore crucial to reduce the overall fault-tolerant implementation cost of a quantum circuit. That is why various methods have been developed to optimize the number of non-Clifford rotation gates, notably for Clifford+R_Z circuits [108–110, 112, 131].

A simple yet efficient strategy to reduce the number of non-Clifford rotation gates in a quantum circuit is to find rotations that can be merged into a single rotation gate [109, 110]. To find the non-Clifford rotation gates which can be merged, it is useful to represent the operation performed by the circuit as a sequence of Pauli rotations followed by a final Clifford operator as explain in Section 1.2.3.4. The number of Pauli rotations in the sequence is then equal to the number of non-Clifford rotation gates in the circuit. If the Pauli products of two Pauli rotations in the sequence are identical and if these Pauli rotations are not separated by another Pauli rotation in the sequence with which they anticommute, then they can be merged into a single Pauli rotation. The complexity of this method, first developed in Reference [110], mainly resides in the number of commutativity checks (i.e. checking whether or not two Pauli rotations commute) that must be performed. In the worst case, the commutativity between each pair of Pauli rotations must be evaluated, leading to $\mathcal{O}(m^2)$ commutativity checks, where m is the number of Pauli rotations. In this chapter, we show how the number of commutativity checks can be greatly reduced. More precisely, we present a method in which the number of commutativity checks is $\mathcal{O}(mh)$ in the worst case, where h satisfies $h < m$ and is the minimal number of internal Hadamard gates required to implement the sequence of Pauli rotations. The number of internal Hadamard gates corresponds to the number of Hadamard gates lying between the first and the last non-Clifford rotation gate of the circuit. An algorithm to implement a sequence of Pauli rotations with a minimal number of internal Hadamard gates was presented in Chapter 6. This algorithm is used in our method to reduce the number of commutativity checks.

An interesting property of this approach is that the structure of the quantum circuit is preserved. Only non-Clifford gates are modified: some are removed, while the angles of others are adjusted. As such, this approach reduces the number of non-Clifford gates in the circuit without increasing the number of Clifford gates. That is why this approach is used to reduce the number of non-Clifford gates in various quantum compilers designed for NISQ devices [132–134]. This way of optimizing the number of T gates is outperformed by some other approaches on some quantum circuits. However, this strategy generally leads to an important reduction in

the number of T gates and its execution is much faster than other approaches. Although this approach does not affect the Clifford gates in the circuit, the reduction of the number of T gates can improve the performances of other algorithms designed for optimizing the number of Clifford gates. For example, we have shown in Chapter 6, that the algorithm presented for reducing the number of internal Hadamard gates achieves much better reductions when the number of T gates have been optimized using this strategy. Reducing the number of internal Hadamard gates is an important pre-processing step to realize before executing some other T-count optimizers, as fewer internal Hadamard gates result in shorter execution times, fewer T gates, and a reduced need for ancillary qubits in the optimized circuit [108, 112, 131].

Then, in Section 7.1, we present an efficient algorithm for optimizing the number of non-Clifford rotation gates and for optimizing the number of parametrized rotation gates in parametrized quantum circuits. We then show that our algorithm is suboptimal for some Clifford+T circuits. We propose a solution to this shortcoming in Section 7.2 which leads to an algorithm with an increased worst-case complexity but that is faster than analogous algorithms and that achieves the same reduction in the number of T gates. Finally, we provide benchmarks in Section 7.3 to evaluate the performances of our algorithms on a library of reversible logic circuits.

7.1 Pauli rotation merging

7.1.1 Problem statement

As explain in Section 1.2.3.4, any unitary gate U, representing the operation performed by a quantum circuit acting on n qubits and composed of Clifford gates and single-qubit rotation gates, can be described by a sequence of Pauli rotations and a final Clifford operator $C \in \mathcal{C}_n$ [21]:

$$U = e^{i\phi} C \left(\prod_{i=1}^{m} R_{P_i}(\theta_i) \right) \tag{7.1}$$

where m is the number of non-Clifford rotation gates in the circuit and P_i is a Pauli product of size n different from $I^{\otimes n}$, for all i.

Consider the sequence of m Pauli products $P_1, \ldots, P_m$ represented in Equation 7.1. If this sequence contains two Pauli products P_i and P_j where i, j are satisfying $1 \leq i < j \leq m$ and such that $P_i = P_j$ and if there exists no k satisfying $i < k < j$ and such that P_k anticommutes with P_i, then the two Pauli rotations associated with P_i and P_j can be merged into a single Pauli rotation. Merging these two Pauli rotations will form a Pauli rotation equal to $R_{P_i}(\theta_i + \theta_j)$. This way of modifying the sequence of Pauli rotations leads to a simple and effective method to reduce the number of Pauli rotations, and therefore the number of non-Clifford rotation gates. This method was first introduced in Reference [110]. An analogous method was independently introduced in Reference [109]. Although the approaches taken in these references are different, it has been proven that they are in fact equivalent [126]. This method of optimizing the number of Pauli rotations leads to the following Pauli rotation merging problem.

Problem 3 (Pauli rotation merging)**.** *Given a sequence of Pauli rotations $R_{P_1}(\theta_1), \ldots, R_{P_m}(\theta_m)$, find all pairs of integers i, j satisfying $1 \leq i < j \leq m$, $P_i = P_j$ and such that there exists no integer k satisfying $i < k < j$ such that P_i anticommutes with P_k.*

In Section 7.1, we present an algorithm that efficiently solves the Pauli rotation merging problem and constructs the associated optimized quantum circuit. For completeness, we first describe the algorithm presented in Reference [110] for optimizing the number of T gates in

Clifford+T circuits. We will use the term `TMerge` to denote this procedure. The input circuit must first be put into the form of Equation 7.1: a sequence of $\pi/4$ Pauli rotations associated with the Pauli products $P_1, \ldots, P_m$ and a final Clifford operator denoted by C_f. Then, the algorithm proceeds as follows. Let U and C be equal to the identity operator. For i going from 1 to m, do the following:

1. Add the Pauli rotation $C^\dagger R_{P_i}(\pi/4)C$ to U, i.e. $U \leftarrow C^\dagger R_{P_i}(\pi/4)CU$.
2. Iterate over all other Pauli rotations $R_Q(\pi/4)$ in U until one of the conditions is true:
   (a) If Q anticommutes with $C^\dagger P_i C$.
   (b) If $Q = C^\dagger P_i C$ then remove Q and $C^\dagger P_i C$ from U and $C \leftarrow CR_{P_i}(\pi/2)$.
   (c) If $Q = -C^\dagger P_i C$ then remove Q and $C^\dagger P_i C$ from U.

At the end, the optimized circuit is obtained by performing the synthesis of the sequence of $\pi/4$ Pauli rotations constituting U and the Clifford operator $C_f C$. Extracting the sequence of Pauli rotations and the final Clifford circuit implemented by the input circuit can be done in $\mathcal{O}(nM)$ where n is the number of qubits and M is the number of gates in the input circuit. The number of commutativity checks performed by this algorithm in the step 2(a) is $\mathcal{O}(m^2)$ in the worst case. Each commutativity check requires $\mathcal{O}(n)$ operations, therefore this algorithm has a complexity of $\mathcal{O}(nM + nm^2)$.

In this section we show how the number of commutativity checks can be greatly reduced in some cases. In Section 7.1.2, we present an algorithm that efficiently solves the Pauli rotation merging problem and constructs the associated optimized quantum circuit with a complexity of $\mathcal{O}(nM + nhm)$ where h is the optimal of internal Hadamard gates required to implement the initial sequence of Pauli rotations. In Section 7.2, we present a variant of the algorithm for optimizing the number of T gates in Clifford+T quantum circuit.

7.1.2 Lowering the number of commutativity checks

In the same way as in Chapter 6, we encode a sequence of m Pauli products into a block matrix $\mathcal{S} = \begin{bmatrix} \mathcal{Z} \\ \mathcal{X} \end{bmatrix}$ of size $2n \times m$ where n is the number of qubits, the submatrix $\mathcal{Z}$ corresponds to the first n rows of $\mathcal{S}$ and the submatrix $\mathcal{X}$ corresponds to the last n rows of $\mathcal{S}$. The value $(\mathcal{Z}_{i,j}, \mathcal{X}_{i,j})$ represents the ith component of the jth Pauli product encoded by $\mathcal{S}$, such that the values $(0,0), (0,1), (1,1)$ and $(1,0)$ are corresponding to the Pauli matrices I, X, Y and Z respectively. We use the notation $\mathcal{S}_{:,i}$ to refer to the $(i+1)$th column of the matrix $\mathcal{S}$ and the notation $\mathcal{S}_{:,:i}$ to refer to the submatrix formed by the first $(i+1)$th columns of $\mathcal{S}$. We will say that $S_{:,i}$ commutes (or anticommutes) with $S_{:,j}$ if the Pauli products that they are respectively encoding are commuting (or anticommuting). The commutativity matrix associated with a sequence of Pauli products encoded in $\mathcal{S}$ will be denoted $A^{(\mathcal{S})}$. For convenience we will drop the superscript $(\mathcal{S})$ from A when it is clear from the context that A is associated with $\mathcal{S}$.

A rank vector $\boldsymbol{v} \in \mathbb{F}_2^m$ associated with a commutativity matrix A satisfies $|\boldsymbol{v}| = \text{rank}(A)$ and is defined as follows.

Definition 4 (Rank vector)**.** The rank vector $\boldsymbol{v} \in \mathbb{F}_2^m$ associated with a commutativity matrix A of size $m \times m$ is defined as follows:

$$v_i = \begin{cases} 0 & \text{if } \text{rank}(A_{:,:i}) = \text{rank}(A_{:,:i-1}), \\ 1 & \text{otherwise,} \end{cases}$$

Algorithm 9: Computes the rank vector associated with a Clifford+R_Z circuit

Input: A Clifford+R_Z circuit C containing m non-Clifford R_Z gates.
Output: The rank vector $\boldsymbol{v}$ associated with C.

```
procedure RankVector(C)
    C' ← InternalHOpt(C)
    v ← vector of size m filled with 0
    for i ← 1 to m − 1 do
        if there is a Hadamard gate between the ith and (i + 1)th R_Z gate of C' then
            v_i ← 1
        end
    end
    return v
```

with $v_0 = 0$.

Let C be a Clifford+R_Z circuit which implements a sequence of Pauli rotations, as described by Equation 7.1, encoded by $\mathcal{S}$. We say that $\boldsymbol{v}$ is the rank vector of C if $\boldsymbol{v}$ is the rank vector associated with the commutativity matrix $A^{(\mathcal{S})}$. The rank vector $\boldsymbol{v}$ of a circuit C can be computed by Algorithm 9. The algorithm starts by calling the `InternalHOpt` procedure presented in Chapter 6. It has been demonstrated that the `InternalHOpt` procedure produces a circuit C' with a minimal number of internal Hadamard gates such that the sequences of Pauli rotations and the final Clifford operators of C and C', as given by Equation 7.1, are identical. Also, the number of internal Hadamard gates in C' is equal to the rank of $A^{(\mathcal{S})}$, which corresponds to the Hamming weight of the rank vector $\boldsymbol{v}$ associated with $\mathcal{S}$, where $\mathcal{S}$ is encoding the Pauli products of the sequence of Pauli rotations implemented by C'. Furthermore, if the last $m - i - 1$, such that i satisfies $0 \leq i < m$, non-Clifford R_Z gates of C' are removed, then C' implements the sequence of Pauli rotations encoded by $S_{:,:i}$ up to a final Clifford circuit with an optimal number of internal Hadamard gates equal to $\text{rank}(A_{:,:i})$. We can then deduce that $v_i = 1$ if and only if there is a Hadamard gate between the ith and $(i+1)$th non-Clifford R_Z gate of C'.

The `InternalHOpt` procedure has a complexity of $\mathcal{O}(nM + n^2h)$ where n is the number of qubits, M is the number of gates in the input circuit and h is the number of internal Hadamard gates in the produced circuit and satisfies $h < m$ where m is the number of non-Clifford R_Z gates in the input circuit. The circuit produced by the `InternalHOpt` procedure contains $\mathcal{O}(nm + n^2)$ gates, therefore the loop in Algorithm 9 can be executed in $\mathcal{O}(nm + n^2)$ operations. Thus, the overall complexity of Algorithm 9 is $\mathcal{O}(nM + n^2h)$.

The following theorem characterizes how the information encoded in the rank vector is valuable for solving the Pauli rotation merging problem (Problem 3) efficiently.

Theorem 11. *Let $\mathcal{S}$ be a sequence of m Pauli products, let A be its commutativity matrix, let $\boldsymbol{v}$ be the rank vector associated with A, and let i and j be integers satisfying $0 \leq i < j < m$. If $\mathcal{S}_{:,i}$ commutes with $\mathcal{S}_{:,k}$ for all integer k satisfying $i < k \leq j$ and $v_k = 1$, then $\mathcal{S}_{:,i}$ commutes with $\mathcal{S}_{:,k}$ for all k satisfying $i < k \leq j$.*

Proof. For all k satisfying $i < k \leq j$ and $v_k = 1$, if $\mathcal{S}_{:,i}$ commutes with $\mathcal{S}_{:,k}$ then we have $A_{i,k} = 0$. Let k be the smallest value satisfying $i < k \leq j$ and $v_k = 0$, then the following equation

$$\bigoplus_{\ell \in L} A_{:,\ell} = A_{:,k} \tag{7.2}$$

is satisfied for some $L \subseteq \{\ell \mid \ell < k\}$. We have $A_{i,\ell} = 0$ by definition if $\ell \leq i$, and if $i < \ell < k$ then $v_\ell = 1$ and so $A_{i,\ell} = 0$ as demonstrated previously. Therefore, for all $\ell \in L$ we have $A_{i,\ell} = 0$ and so by Equation 7.2 we necessarily have $A_{i,k} = 0$ which means that $\mathcal{S}_{:,i}$ commutes with $\mathcal{S}_{:,k}$. It follows that $\mathcal{S}_{:,i}$ commutes with $\mathcal{S}_{:,k}$ for all k satisfying $i < k \leq j$. □

Algorithm 10: Black box rotation merging algorithm

Input: A Clifford+R_Z circuit C containing m non-Clifford R_Z gates.
Output: An optimized Clifford+R_Z circuit equivalent to C.

```
1  procedure BBMerge(C)
2      v ← RankVector(C)
3      r ← new empty vector
4      T ← new tableau
5      S ← new empty sequence of Pauli products
6      D ← new empty associative array
7      t ← 0
8      foreach gate G ∈ C do
9          if G is Clifford then
10             Prepend G† to T
11         else if G is a non-Clifford R_Z(θ) gate acting on qubit i then
12             r_t ← θ
13             P ← ith stabilizer generator of T
14             Append P to S
15             Append t to D[P]
16             if the list D[P] contains a value not equal to t then
17                 merge ← true
18                 j ← last value of the list D[P] not equal to t
19                 for k ← j to t − 1 do
20                     if v_k = 1 and P anticommutes with S_{:,k} then
21                         merge ← false
22                         break
23                 end
24                 if merge is true then
25                     r_t ← r_j + r_t
26                     r_j ← 0;
27                     Remove j from D[P]
28                     if r_t ≡ 0 (mod π/2) then
29                         Prepend the R_Z^†(r_t) gate acting on qubit i to T
30                         Remove t from D[P]
31             t ← t + 1
32     end
33     return C where the ith non-Clifford R_Z(θ) gate is replaced by R_Z(r_i) for all i
```

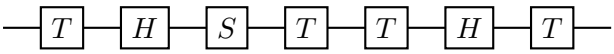
Consider the algorithm whose pseudocode is presented in Algorithm 10, and which takes a Clifford+R_Z circuit C as input. The vector $\boldsymbol{r}$ encodes the angles of the sequence of Pauli rotations associated with C such that the ith Pauli rotation have an angle equal to r_i. A tableau $\mathcal{T}$ is a matrix composed of $4n^2 + 2n$ bits encoding the operation performed by a Clifford operator

in $\mathcal{C}_n$ [17]. The matrix $\mathcal{S}$ is encoding a sequence of Pauli products, as previously defined. The variable $\mathcal{D}$ is an associative array which maps a Pauli product to a list of integers, we will use $\mathcal{D}[P]$ to refer to the list of integers associated with the Pauli product P. The variable t is a counter for the number of non-Clifford R_Z gates that have been processed so far by the algorithm.

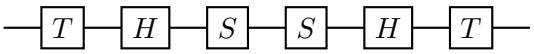
The main loop of the algorithm iterates over the gates of the input circuit C. If the gate G is Clifford then $G^\dagger$ is prepended to the tableau $\mathcal{T}$, so that $\mathcal{T}$ is tracking the inverse of the circuit processed so far without the non-Clifford R_Z gates. Otherwise, if the gate G is not Clifford then it must be a non-Clifford $R_Z(\theta)$ gate acting on some qubit i. The Pauli rotation associated with this $R_Z(\theta)$ gate is $\tilde{C}^\dagger R_Z(\theta)\tilde{C}$ where $\tilde{C}$ is the Clifford operator obtained from the input circuit C by truncating it after this $R_Z(\theta)$ gate and by removing all its non-Clifford gates. The Clifford operator $\tilde{C}^\dagger$ is encoded by the tableau $\mathcal{T}$, and therefore the Pauli product P associated with the Pauli rotation $R_P(\theta) = \tilde{C}^\dagger R_Z(\theta)\tilde{C}$ is given by the ith stabilizer generator of $\mathcal{T}$. The algorithm then determines whether or not there is a potential Pauli rotation which can be merged with $R_P(\theta)$ by checking if the list $\mathcal{D}[P]$ contains the index associated with another Pauli rotation. If this is the case, then the corresponding index j is retrieved from the list $\mathcal{D}[P]$. The value j corresponds to the jth Pauli rotation of C when it is put into the form of Equation 7.1, which is encoded by $\mathcal{S}_{:,j}$. We can rely on Theorem 11 to determine whether or not the Pauli rotations associated with $\mathcal{S}_{:,j}$ and P can be merged. If the condition of Theorem 11 is not satisfied, then the Pauli rotations will not be merged, and t will remain on the list $\mathcal{D}[P]$ for a potential merge with a Pauli rotation that has not yet been processed. Otherwise, if the condition is satisfied then the two Pauli rotations can be merged, the angle of the jth Pauli rotation encoded by r_j is set to 0, and the angle of the Pauli rotation associated with P which is encoded by r_t is set to $r_j + r_t$. Then j must be removed from $\mathcal{D}[P]$ as the jth Pauli rotation has been merged, and the variable t is updated. In the case where $r_j + r_t$ is a multiple of $\pi/2$, then t must be removed from $\mathcal{D}[P]$ as well because it is not associated with a non-Clifford Pauli rotation anymore. When all gates within C have been processed, the optimized circuit is obtained by replacing the ith non-Clifford $R_Z(\theta_i)$ gate from C with $R_Z(r_i)$ for all i.

Complexity analysis. The rank vector $\boldsymbol{v}$ can be computed by Algorithm 9 which have a complexity of $\mathcal{O}(nM + n^2h)$ where M is the number of gates in C, n is the number of qubits and h is the minimal number of internal Hadamard gates required to implement the sequence of Pauli rotations associated with C. The main loop of Algorithm 10 performs M iterations. Prepending a gate to the tableau $\mathcal{T}$ can be done in $\mathcal{O}(n)$ operations. The second condition of the main loop (if G is a non-Clifford $R_Z(\theta)$ gate) is evaluated to true m times where m is the number of non-Clifford $R_Z(\theta)$ gates in C. Each operation of the associative array $\mathcal{D}$ can be done with a complexity of $\mathcal{O}(n)$. Verifying the condition of Theorem 11 induces a cost of $\mathcal{O}(nh)$ because $|\boldsymbol{v}| = h$, and so at most $\mathcal{O}(h)$ commutativity checks are performed and a commutativity check can be done in $\mathcal{O}(n)$ operations. The loop on line 19 of Algorithm 10 performs at most m iterations and is executed $\mathcal{O}(m)$ times, leading to a complexity of $\mathcal{O}(m^2)$. However this can be reduced to a complexity of $\mathcal{O}(hm)$ by using a vector storing the h indices for which $v_i = 1$ instead on using $\boldsymbol{v}$. All the operations performed when two Pauli rotations are merged can be done with a complexity of $\mathcal{O}(n)$. Thus, the overall complexity of Algorithm 10 is $\mathcal{O}(nM + n^2h + nhm)$, which corresponds to a complexity of $\mathcal{O}(nM + nhm)$ in the typical case where $n < m$.

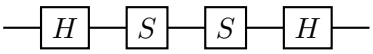
This algorithm have a better complexity than the $\mathcal{O}(nM+nm^2)$ complexity of the `TMerge` procedure of Reference [110] because $h < m$. However, in the worst case we have $h = m-1$, therefore this algorithm and the `TMerge` procedure have the same worst-case complexity of $\mathcal{O}(nM+nm^2)$. Nonetheless, for some circuits h is much smaller than m, which makes Algorithm 10 significantly faster than the `TMerge` algorithm. However, although this algorithm leads to the same T-count



(a) Initial circuit.



(b) Circuit resulting from the execution of the `BBMerge` procedure.



(c) Circuit resulting from the execution of the `FastTMerge` procedure.

Figure 7.1: Example of a circuit for which the `BBMerge` procedure, presented in Algorithm 10, is suboptimal. The initial circuit contains 4 T gates (a), the `BBMerge` procedure produces a circuit containing 2 T gates (b), and the `FastTMerge` procedure, presented in Algorithm 11, produces a circuit that doesn't contain any T gate (c).

reduction as the `TMerge` algorithm for most Clifford+T circuits evaluated in the benchmarks of Section 7.3, it is not always the case for every circuit. A simple example where Algorithm 10, called the `BBMerge` procedure, does not find all the Pauli rotations that can be merged is given in Figure 7.1. This suboptimality is due to the fact that merging two $\pi/4$ Pauli rotations generates a Clifford $\pi/2$ Pauli rotation which modifies the sequence of Pauli rotations. The rank vector $\boldsymbol{v}$ should be updated to take into account these modifications of the sequence of Pauli rotations. However, this is not done by Algorithm 10 as updating the rank vector everytime two Pauli rotations are merged would induce a much more important worst-case complexity. As such, the `BBMerge` procedure finds all possible reductions in the number of non-Clifford R_Z gates using the Pauli rotation merging approach when the angles are black boxes, but it can fail when merging Pauli rotations generates Clifford Pauli rotations. In the next subsection we propose some modifications to our approach in order to take advantage of the reduced number of commutativity checks of Algorithm 10 while obtaining the same T-count reduction as the `TMerge` procedure.

7.2 Extension for Clifford+T circuits

In this section, we present a modified version of Algorithm 10, whose pseudocode is given in Algorithm 11. This algorithm achieves the same T-count reduction as the `TMerge` procedure while taking advantage of Theorem 11 to lower the number of commutativity checks. Algorithm 11 uses the same data structures as Algorithm 10 but with an additional vector $\boldsymbol{w}$ which is initialized as a copy of the rank vector $\boldsymbol{v}$. Let $\mathcal{S}$ be the sequence of Pauli products constructed by Algorithm 10 (and by Algorithm 11 analogously). As previously, we will use $\mathcal{S}_{:,i}$ to refer to the ith Pauli products encoded by the sequence $\mathcal{S}$. As shown by the example of Figure 7.1, Algorithm 10 will not merge two Pauli rotations associated with $\mathcal{S}_{:,i}$ and $\mathcal{S}_{:,j}$ such that $i < j$ and $\mathcal{S}_{:,i} = \mathcal{S}_{:,j}$ when there exists an integer k satisfying $i < k < j$, $v_k = 1$ and $r_k \equiv 0 \pmod{\pi/2}$. Yet, if $r_k \equiv 0 \pmod{\pi/2}$ then the kth Pauli rotation has already been merged with another Pauli rotation to form a Clifford Pauli rotation, and should not prevent the merge of the Pauli rotations associated with $\mathcal{S}_{:,i}$ and $\mathcal{S}_{:,j}$. This problem could be solved by updating the rank vector $\boldsymbol{v}$ using Algorithm 9. However, doing this everytime two Pauli rotations are merged would greatly increase the complexity of the algorithm. Instead, when two Pauli rotations are merged, Algorithm 11 updates the vector $\boldsymbol{w}$ such that two Pauli rotations associated with $\mathcal{S}_{:,i}$ and $\mathcal{S}_{:,j}$ where $\mathcal{S}_{:,i} = \mathcal{S}_{:,j}$ and $i < j$ can be merged if and only if $\mathcal{S}_{:,i}$ commutes with $\mathcal{S}_{:,k}$ for all k satisfying $i < k < j$, $w_k = 1$ and $r_k \not\equiv 0 \pmod{\pi/2}$. The vector $\boldsymbol{w}$ is simply updated by setting w_k to 1

Algorithm 11: Non-Clifford rotation merging algorithm

Input: A Clifford+R_Z circuit C containing m non-Clifford R_Z gates.
Output: An optimized Clifford+R_Z circuit equivalent to C.

procedure `FastTMerge`(C)
  $\boldsymbol{v} \leftarrow$ `RankVector`(C)
  $\boldsymbol{w} \leftarrow$ copy of $\boldsymbol{v}$
  $\boldsymbol{r} \leftarrow$ new empty vector
  $\mathcal{T} \leftarrow$ new tableau
  $\mathcal{S} \leftarrow$ new empty sequence of Pauli products
  $\mathcal{D} \leftarrow$ new empty associative array
  $t \leftarrow 0$
  foreach gate $G \in C$ **do**
    if G is Clifford **then**
      Prepend $G^\dagger$ to $\mathcal{T}$
    else if G is a non-Clifford $R_Z(\theta)$ gate acting on qubit i **then**
      $r_t \leftarrow \theta$
      $P \leftarrow i$th stabilizer generator of $\mathcal{T}$
      Append P to $\mathcal{S}$ and append t to $\mathcal{D}[P]$
      if the list $\mathcal{D}[P]$ contains a value not equal to t **then**
        merge $\leftarrow$ `true`
        $j \leftarrow$ last value of the list $\mathcal{D}[P]$ not equal to t
        for $k \leftarrow j$ **to** $t-1$ **do**
          if $v_k = 1$ **and** P anticommutes with $\mathcal{S}_{:,k}$ **then**
            if $r_k \equiv 0 \pmod{\frac{\pi}{2}}$ **then**
              for $\ell \leftarrow k+1$ **to** $t-1$ **do**
                if $w_\ell = 1$, $r_\ell \not\equiv 0 \pmod{\frac{\pi}{2}}$ **and** P anticommutes with $\mathcal{S}_{:,\ell}$ **then**
                  merge $\leftarrow$ `false`
                  `break`
              end
            else
              merge $\leftarrow$ `false`
            `break`
        end
        if *merge* is `true` **then**
          if $v_j = 1$ **then**
            for $k \leftarrow j+1$ **to** $t-1$ **do** $w_k \leftarrow 1$;
            $w_j = 0$
          $r_t \leftarrow r_t + r_j$
          $r_j \leftarrow 0$
          Remove j from $\mathcal{D}[P]$
          if $r_t \equiv 0 \pmod{\frac{\pi}{2}}$ **then**
            Prepend the $R_Z^\dagger(r_t)$ gate acting on qubit i to $\mathcal{T}$
            Remove t from $\mathcal{D}[P]$
      $t \leftarrow t+1$
  end
  return C where the ith non-Clifford $R_Z(\theta)$ gate is replaced by $R_Z(r_i)$ for all i

for all k satisfying $i \leq k \leq j$ when the ith Pauli rotation is merged with the jth Pauli rotation. We can summarize this process performed by Algorithm 11 by the following distinction of cases where $i < j$ and $\mathcal{S}_{:,i} = \mathcal{S}_{:,j}$:

- If $\mathcal{S}_{:,i}$ commutes with $\mathcal{S}_{:,k}$ for all k satisfying $i < k < j$ and $v_k = 1$, then, as stated by Theorem 11, the Pauli rotations associated with $\mathcal{S}_{:,i}$ and $\mathcal{S}_{:,j}$ can be merged in the same way as in Algorithm 10.

- If $\mathcal{S}_{:,i}$ anticommutes with $\mathcal{S}_{:,k}$ for some k satisfying $i < k < j$, $v_k = 1$ and $r_k \not\equiv 0 \pmod{\pi/2}$, then the Pauli rotations associated with $\mathcal{S}_{:,i}$ and $\mathcal{S}_{:,j}$ cannot be merged.

- If $\mathcal{S}_{:,i}$ anticommutes with $\mathcal{S}_{:,k}$ for some k satisfying $i < k < j$, $v_k = 1$ and $r_k \equiv 0 \pmod{\pi/2}$, then the algorithm will check whether or not $\mathcal{S}_{:,j}$ anticommutes with $\mathcal{S}_{:,\ell}$ for some ℓ satisfying $k < \ell < j$, $w_\ell = 1$ and $r_k \not\equiv 0 \pmod{\pi/2}$. If this condition is evaluated to true, then the Pauli rotations associated with $\mathcal{S}_{:,i}$ and $\mathcal{S}_{:,j}$ cannot be merged.

This algorithm is then finding all Pauli rotations that can be merged as the sequence of Pauli products $\mathcal{S}$ associated with the resulting sequence of Pauli rotations cannot contain $\mathcal{S}_{:,i}$ and $\mathcal{S}_{:,j}$ such that $i < j$, $\mathcal{S}_{:,i} = \mathcal{S}_{:,j}$ and $\mathcal{S}_{:,i}$ commutes with $\mathcal{S}_{:,k}$ for all k satisfying $i < k < j$.

Complexity analysis. Let n be the number of qubits, m be the number of non-Clifford R_Z gates in the initial circuit and M be the number of gates in the initial circuit. When compared to Algorithm 10, Algorithm 11 is composed of an additional loop (in line 23) which is called at most m times and performs $\mathcal{O}(m)$ iterations. At each iteration a commutativity check requiring $\mathcal{O}(n)$ operations is performed, therefore this loop induces $\mathcal{O}(nm^2)$ operations. Algorithm 11 also has another additional loop (in line 34) to update the vector $\boldsymbol{w}$. This loop induces a complexity of $\mathcal{O}(hm)$ as it performs $\mathcal{O}(m)$ iterations, each iteration performs a constant number of operations and the loop is called $\mathcal{O}(h)$ times, where h is the minimal number of internal Hadamard gates required to implement the sequence of Pauli rotations associated with the input circuit. Thus, the worst-case complexity of Algorithm 11 is $\mathcal{O}(nM + n^2h + nm^2)$, or $\mathcal{O}(nM + nm^2)$ in the typical case where $n < m$.

This complexity matches the complexity of the `TMerge` procedure of Reference [110] and is worse than the $\mathcal{O}(nM + nhm)$ complexity of Algorithm 10 as $h < m$. However, this algorithm can deal with the specific cases not handled by Algorithm 10, such as the one presented in Figure 7.1. As such, it reduces the number of non-Clifford R_Z gates as much as possible by using the Pauli rotation merging approach while trying to take advantage of Theorem 11 to be more efficient by reducing the number of commutativity checks. We present benchmarks in Section 7.3 to demonstrate the effectiveness of Algorithm 10 and Algorithm 11.

7.3 Benchmarks

In this section we evaluate the performances of the algorithms we presented, and we compare them to state-of-the-art alternatives. The benchmarks are performed over a set of Clifford+T circuits obtained from References [127, 128]. We also include a circuit implementing the block cipher DEFAULT, as given in Reference [129]. We compare the performances of the algorithm presented in Reference [109], the `BBMerge` procedure (Algorithm 10), the `FastTMerge` procedure (Algorithm 11) and the `TMerge` procedure of Reference [110]. The `BBMerge`, `FastTMerge` and `TMerge` procedures were implemented with the Rust programming language, while the algorithm

		PyZX [109]		BBMerge		FastTMerge		TMerge [110]	
Circuit	T-count	T-count	t (s)	T-count	t (s)	T-count	t (s)	T-count	t (s)
Adder_8	399	173	0	**179**	0	173	0	173	0
Barenco Tof_3	28	16	0	16	0	16	0	16	0
Barenco Tof_4	56	28	0	28	0	28	0	28	0
Barenco Tof_5	84	40	0	40	0	40	0	40	0
Barenco Tof_{10}	224	100	0	100	0	100	0	100	0
CSLA MUX_3	70	62	0	62	0	62	0	62	0
CSUM MUX_9	196	84	0	84	0	84	0	84	0
DEFAULT	62720	-	-	39744	6	39744	6	39744	8
GF(2^4) Mult	112	68	0	68	0	68	0	68	0
GF(2^5) Mult	175	115	0	115	0	115	0	115	0
GF(2^6) Mult	252	150	0	150	0	150	0	150	0
GF(2^7) Mult	343	217	0	217	0	217	0	217	0
GF(2^8) Mult	448	264	0	264	0	264	0	264	0
GF(2^9) Mult	567	351	0	351	0	351	0	351	0
GF(2^{10}) Mult	700	410	1	410	0	410	0	410	0
GF(2^{16}) Mult	1792	1040	4	1040	0	1040	0	1040	0
GF(2^{32}) Mult	7168	4128	47	4128	0	4128	0	4128	6
GF(2^{64}) Mult	28672	16448	423	16448	0	16448	0	16448	99
GF(2^{128}) Mult	114688	65664	8917	65664	2	65664	2	65664	1930
GF(2^{256}) Mult	458752	-	-	262400	18	262400	18	262400	42304
GF(2^{512}) Mult	1835008	-	-	1048576	137	1048576	138	-	-
Grover_5	336	166	0	166	0	166	0	166	0
Ham_{15} (high)	2457	1019	9	**1021**	0	1019	0	1019	0
Ham_{15} (low)	161	97	0	97	0	97	0	97	0
Ham_{15} (med)	574	212	2	**242**	0	212	0	212	0
HWB_6	105	75	0	75	0	75	0	75	0
HWB_8	5887	3517	43	**3583**	0	3517	0	3517	0
HWB_{10}	29939	15891	910	**16371**	0	15891	0	15891	1
HWB_{11}	84196	44500	7473	**45610**	1	44500	1	44500	1
HWB_{12}	171465	85611	52910	**87035**	2	85611	2	85611	5
Mod Adder_{1024}	1995	1011	3	1011	0	1011	0	1011	0
Mod Mult_{55}	49	35	0	35	0	35	0	35	0
Mod Red_{21}	119	73	0	73	0	73	0	73	0
Mod5_4	28	8	0	8	0	8	0	8	0
QCLA Adder_{10}	238	162	0	162	0	162	0	162	0
QCLA Com_7	203	95	0	95	0	95	0	95	0
QCLA Mod_7	413	237	0	237	0	237	0	237	0
QFT_4	69	67	0	67	0	67	0	67	0
RC Adder_6	77	47	0	47	0	47	0	47	0
Tof_3	21	15	0	15	0	15	0	15	0
Tof_4	36	23	0	23	0	23	0	23	0
Tof_5	49	31	0	31	0	31	0	31	0
Tof_{10}	119	71	0	71	0	71	0	71	0
VBE Adder_3	70	24	0	24	0	24	0	24	0

Table 7.1: Comparison of different procedures for optimizing the number of T gates in Clifford+T circuits. The first T-count column indicates the number of T gates in the initial circuits. The T-count after optimization and the execution time in seconds is reported for each procedure. A blank entry indicates that the execution couldn't be carried out in less than a day. A T-count entry is in bold if the `BBMerge` procedure didn't achieve the same T-count as the other methods.

of Reference [109] was implemented in Python in the PyZX library [135]. Our implementations of Algorithm 10 and Algorithm 11 used for the benchmarks are open source [8].

The results of our benchmarks are presented in Table 7.1. As expected, the `FastTMerge` procedure and the algorithms presented in Reference [110] and Reference [109] all obtain the same reduction in the number of T gates. The `BBMerge` procedure has the lowest execution time but it fails to achieve the same number of T gates as the other algorithms on a few circuits. This indicates that, for these circuits, if the non-Clifford rotation gates are treated as black boxes, as done by the `BBMerge` procedure, it is not possible to find all the non-Clifford rotation gates that can be merged into a single rotation gate. To achieve better optimization, as done by the other algorithms, it is necessary to know the angles of the non-Clifford rotation gates.

We can notice that the `BBMerge` and `FastTMerge` procedures have very similar execution times, despite not having the same complexity. Also, on large circuits, these procedures have much lower execution times than the algorithms presented in References [109, 110]. For instance, the "GF(2^{512}) Mult" circuit can be optimized in less than 3 minutes by the `BBMerge` and `FastTMerge` procedures, whereas the other algorithms could not optimize this circuit whithin a day. The `BBMerge` and `FastTMerge` procedures are particularly efficient on the "GF(2^n) Mult" circuits because these circuits can be rewritten without any internal Hadamard gates. Thus, by computing the rank vector, the procedures will detect that all the Pauli rotations associated with the circuit are commuting, and so no commutativity check will have to be performed. This best-case scenario for the `BBMerge` and `FastTMerge` procedures is, on the contrary, the worst-case scenario for the `TMerge` procedure of Reference [110]. Indeed, in such cases, the `TMerge` procedure will have to perform $\mathcal{O}(m^2)$ commutativity checks, where m is the number of T gates in the initial circuit.

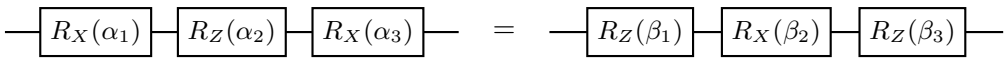
The worst-case complexity of the `BBMerge` procedure is $\mathcal{O}(nM+nhm)$, whereas the worst-case complexity of the `FastTMerge` procedure is $\mathcal{O}(nM + nm^2)$, where n is the number of qubits, M is the number of gates in the initial circuit, m is the number of T gates in the initial circuit, and h satisfies $h < m$ and is the minimal number of internal Hadamard gates required to implement the sequence of Pauli rotations associated with the initial circuit. Thus, an open problem is to bridge the complexity gap between the `BBMerge` procedure and the `FastTMerge` procedure. Does there exist an algorithm that achieves the same T-count reduction as the `FastTMerge` procedure but with the same time complexity as the `BBMerge` procedure? If such an algorithm exists, then, without increasing the time complexity, it could be coupled with the algorithm presented in Chapter 6, which finds the optimal number of Hadamard gates and internal Hadamard gates in the circuit as long as the associated sequence of Pauli rotations is not altered. This would result in a particularly efficient procedure for optimizing both the number of T gates and the number of Hadamard gates, which can be used as a pre-processing step to improve the efficiency of more advanced T-count optimizers [108, 112, 131].

Chapter 8

Optimality in Parametrized Clifford Circuits with Non-Repeated Parameters

Parametrized quantum circuits are quantum circuits that include adjustable parameters in their quantum gates. They are central to variational quantum algorithms such as QAOA [136] or VQE [137]. In such circuits, the parameterized rotation gates are often non-Clifford gates, making their optimization crucial for improving the efficiency of these algorithms.

In this chapter, we prove that Algorithm 10, presented in Chapter 7, produces a circuit with a minimal number of parametrized rotation gates when the input circuit is composed of Clifford gates and parametrized rotation gates, and when each parameter can only appear once in the circuit. Our results mainly differ from Reference [138] in the parameter transformations considered between equivalent parametrized quantum circuits. The equivalence between parametrized quantum circuits considered in Reference [138] is restricted to cases where the relations between parameters are analytic functions. In this work, we do not impose such a restriction on the relations between parameters. A consequence of the restricted parameter transformations considered in Reference [138] is that the following Euler angle transformation rewriting rule:



(8.1)

where $\beta_1, \beta_2, \beta_3$ are dependent on $\alpha_1, \alpha_2, \alpha_3$ through some trigonometric relations, is excluded as it involves discontinuous parameter transformations. Our results show that this rewriting rule is not useful for finding an equivalent parametrized Clifford circuit with an optimal number of parametrized rotation gates and non-repeated parameters.

On the basis of this result, we also prove that the algorithm presented in Chapter 6 to optimize the number of Hadamard gates and internal Hadamard gates is optimal for parametrized Clifford circuits with non-repeated parameters. For parametrized circuits composed of Clifford gates and parametrized rotation gates, the number of internal Hadamard gates corresponds to the number of Hadamard gate lying between the first and the last parametrized rotation gate of the circuit. Combined with our algorithm designed to optimize the number of parametrized rotation gates, this leads to a procedure for achieving an optimal number of parametrized rotation gates, Hadamard gates, and internal Hadamard gates, with complexity of $\mathcal{O}(nM + nhm)$, where n is the number of qubits, M is the number of gates in the initial circuit, m is the initial number of parametrized rotation gates and h is the optimal number of internal Hadamard gates.

In Section 8.1, we introduce parametrized Clifford circuits along with related definitions. Then, in Section 8.2, we prove the optimality of Algorithm 10 for optimizing the number of parametrized rotations in parametrized Clifford circuits with non-repeated parameters. On the basis of this result, in Section 8.3, we prove that the Hadamard gate optimization algorithm presented in Chapter 6 is optimal in the number of Hadamard gates and internal Hadamard gates for parametrized Clifford circuits with non-repeated parameters.

8.1 Parametrized Clifford circuits

We are interested in Clifford+R_Z parametrized quantum circuits that do not have non-Clifford constant rotation gates. Such parametrized quantum circuits are called parametrized Clifford circuits and are defined as follows.

Definition 5 (Parametrized Clifford circuit)**.** A parametrized Clifford circuit with parameters $\boldsymbol{\alpha} \in \mathbb{R}^\kappa$ is a quantum circuit composed of Clifford gates and parametrized rotation gates $R_Z(f_1(\boldsymbol{\alpha}))$, ..., $R_Z(f_m(\boldsymbol{\alpha}))$, where f_i is any non-constant function satisfying $f_i(\mathbf{0}) = 0$ and $f_i(\boldsymbol{\alpha}) \in [0, 2\pi)$ for all $\boldsymbol{\alpha}$.

As explained in Section 1.2.3.4 for Clifford+R_Z circuits, the parametrized unitary gate U associated with a parametrized Clifford circuit acting on n qubits and composed of parametrized rotation gates $R_Z(f_1(\boldsymbol{\alpha})), \ldots, R_Z(f_m(\boldsymbol{\alpha}))$ can be represented by a sequence of parametrized Pauli rotations and a Clifford operator:

$$U(\boldsymbol{\alpha}) = e^{i\phi} C \left(\prod_{i=1}^{m} R_{P_i}(f_i(\boldsymbol{\alpha})) \right) \tag{8.2}$$

where $P_i \in \mathcal{P}_n \setminus \{\pm I^{\otimes n}\}$, C is a Clifford operator, and assuming that either $R_Z(f_i(\boldsymbol{\alpha}))$ appears after $R_Z(f_{i+1}(\boldsymbol{\alpha}))$ in the circuit, or that the two gates are parallel in the circuit. Note that the Pauli rotations in Equation 8.2 are not necessarily non-Clifford as we can have Clifford parametrized rotation gates in the circuit. A sequence of Pauli rotations associated with a parametrized Clifford circuit refers to a sequence of Pauli rotations obtained when the circuit is put into the form of Equation 8.2.

Definition 6. Given $S \subseteq [1..\kappa]$, let $p_S : \mathbb{R}^\kappa \to \mathbb{R}^\kappa$ be the projector on S defined as, $\forall \boldsymbol{\alpha} \in \mathbb{R}^\kappa, \forall i \in [1..\kappa]$,

$$(p_S(\boldsymbol{\alpha}))_i = \begin{cases} \boldsymbol{\alpha}_i & \text{if } i \in S, \\ 0 & \text{otherwise.} \end{cases} \tag{8.3}$$

As a function may not depend on some of its parameters, we define its useful parameters as the parameters that actually have an effect on the value of a function.

Definition 7 (Useful parameters)**.** The useful parameters of a function $f : \mathbb{R}^\kappa \to \mathbb{R}$ is the smallest[1] set $\mathrm{par}(f)$ such that $f \circ p_{\mathrm{par}(f)} = f$.

Note that the functions in a parametrized Clifford circuit always have a non-empty set of useful parameters as they are non-constant. We will also be interested in particular subsets of the useful parameters of a function, referred to as minimal supports and defined as follows.

[1]such set is unique as $f \circ p_S = f$ and $f \circ p_{S'} = f$ imply that $f \circ p_{S \cap S'} = f$.

Definition 8 (Minimal support)**.** A *minimal support* of a function $f : \mathbb{R}^\kappa \to \mathbb{R}$ is a set $S \subseteq [1..\kappa]$ such that $f \circ p_S \neq 0$ and $\forall T \subset S$, $f \circ p_T = 0$.

For example, the minimal supports of the function $f(\alpha_1, \alpha_2) = \alpha_1 + \alpha_2$ are $\{1\}$ and $\{2\}$. Whereas the only minimal support of the function $f(\alpha_1, \alpha_2) = \alpha_1\alpha_2$ is $\{1, 2\}$. Note that a non-constant function always have at least one minimal support. This notion of minimal support will be useful for the study of parametrized Clifford circuit with non-repeated parameters, defined as follows.

Definition 9 (Parametrized Clifford circuit with non-repeated parameters)**.** A parametrized Clifford circuit with non-repeated parameters is a parametrized Clifford circuit with parameters $\boldsymbol{\alpha} \in \mathbb{R}^\kappa$ and parametrized rotation gates $R_Z(f_1(\boldsymbol{\alpha})), \ldots, R_Z(f_m(\boldsymbol{\alpha}))$ where f_i and f_j have disjoint sets of useful parameters for all integers i, j satisfying $1 \leq i < j \leq m$.

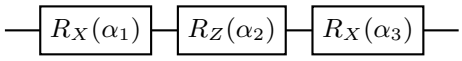
In Section 8.2, we prove that Algorithm 10, prsented in Chapter 7, is optimally minimizing the number of parametrized rotation gates in parametrized Clifford circuit with non-repeated parameters. Then, in Section 8.3, we prove that the algorithm presented in Chapter 6 is optimally minimizing the number of Hadamard gates and internal Hadamard gates in parametrized Clifford circuit with non-repeated parameters.

8.2 Optimality in the number of parametrized rotations

In this section, we prove the following theorem, which states the optimality of Algorithm 10 for reducing the number of parametrized rotations in a parametrized Clifford circuit with non-repeated parameters.

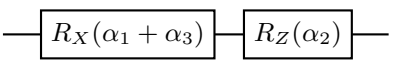
Theorem 12. *Let C_o be the circuit produced by Algorithm 10 when a parametrized Clifford circuit with non-repeated parameters is given as input. Then C_o is a parametrized Clifford circuit with non-repeated, and the number of parametrized rotations in C_o is optimal: there does not exist a parametrized Clifford circuit with non-repeated parameters equivalent to C_o which contains a lower number of parametrized rotations.*

Note that the commutativity checks perform in Algorithm 10 are up to a global phase. This is important for parametrized Clifford circuit because it allows some optimizations. For instance, the following parametrized Clifford circuit



(8.4)

where $\alpha_1, \alpha_2, \alpha_3 \in \{0, \pi\}^3$, can be rewritten, up to a global phase, as



(8.5)

which reduces the number of parametrized rotations. Two Pauli rotations $R_P(\alpha_1)$ and $R_{P'}(\alpha_2)$, where $\alpha_1 \neq 0 \pmod{2\pi}$ and $\alpha_2 \neq 0 \pmod{2\pi}$, commute, up to a global phase, if and only if P commutes with P' or $\alpha_1 \equiv \alpha_2 \equiv \pi \pmod{2\pi}$. Therefore, to execute Algorithm 10 on parametrized Clifford circuits, we need to determine if a parametrized rotation gate with angle α satisfies $\alpha = 0 \pmod{\pi}$. That is the only knowledge required regarding the angles of the parametrized rotation gates.

Theorem 12 is proven by the following lemma, which states that the number of parametrized rotations in the parametrized Clifford circuit with non-repeated parameters produced by Algorithm 2 is optimal, and that all optimal circuits share the same associated sequence of Pauli rotations up to some commutative relations.

Lemma 2. *Let C_o be a parametrized Clifford circuit with non-repeated parameters produced by Algorithm 10. Let C' be a parametrized Clifford circuit with non-repeated parameters equivalent to C_o and which has an optimal number of parametrized rotations. And let $R_{P_1}(f_1(\boldsymbol{\alpha})), \ldots, R_{P_m}(f_m(\boldsymbol{\alpha}))$ and $R_{P'_1}(f'_1(\boldsymbol{\alpha})), \ldots, R_{P'_{m'}}(f'_{m'}(\boldsymbol{\alpha}))$ be sequences of Pauli rotations associated with C_o and C' respectively. Then $m = m'$ and there exists a permutation σ satisfying $R_{P_i}(f_i(\boldsymbol{\alpha})) = R_{P'_{\sigma(i)}}(f'_{\sigma(i)}(\boldsymbol{\alpha}))$ for all i and such that $R_{P'_{\sigma(i)}}(f'_{\sigma(i)}(\boldsymbol{\alpha}))$ commutes, up to a global phase, with $R_{P'_j}(f'_j(\boldsymbol{\alpha}))$ for all $\boldsymbol{\alpha}$ and for all integer j satisfying $\sigma(i) \leq j \leq i$ and $i \leq j \leq \sigma(i)$.*

Proof. We start the proof of Lemma 2 by showing that, for all $\boldsymbol{\alpha}$, $R_{P_1}(f_1(\boldsymbol{\alpha})) = R_{P'_i}(f'_i(\boldsymbol{\alpha}))$ for some integer i, and that $R_{P'_i}(f'_i(\boldsymbol{\alpha}))$ commutes, up to a global phase, with $R_{P'_j}(f'_j(\boldsymbol{\alpha}))$ for all j satisfying $1 \leq j \leq i$. Our proof will rely on the following equality, derived from Equation 8.2:

$$\prod_{k=1}^{m} R_{P_k}(f_k(\boldsymbol{\alpha})) = \prod_{k=1}^{m'} R_{P'_k}(f'_k(\boldsymbol{\alpha})) \tag{8.6}$$

which holds up to a global phase for all $\boldsymbol{\alpha}$ because the parametrized circuits C_o and C' are equivalent. Let S be a minimal support of f_1 and let i be an integer such that $f'_i(p_S(\boldsymbol{\alpha})) \neq 0$ for some $\boldsymbol{\alpha}$. Note that such a function f'_i necessarily exists, otherwise, because S is a minimal support of f_1 and all the f functions have disjoint sets of useful parameters, we would have

$$\prod_{k=1}^{m} R_{P_k}(f_k(p_S(\boldsymbol{\alpha}))) = R_{P_1}(f_1(p_S(\boldsymbol{\alpha}))) \neq I \tag{8.7}$$

and

$$\prod_{k=1}^{m'} R_{P'_k}(f'_k(p_S(\boldsymbol{\alpha}))) = I \tag{8.8}$$

for some $\boldsymbol{\alpha}$, which contradicts Equation 8.6.

We first prove that

$$\prod_{k=1}^{m'} R_{P'_k}(f'_k(p_S(\boldsymbol{\alpha}))) = R_{P'_i}(f'_i(p_S(\boldsymbol{\alpha}))) \tag{8.9}$$

for all $\boldsymbol{\alpha}$. Let's assume that there exist an integer j and a vector $\boldsymbol{\alpha}$ such that $j \neq i$ and $f'_j(p_S(\boldsymbol{\alpha})) \neq 0$. The set S is not a minimal support of f'_j because there exists $\boldsymbol{\alpha}$ such that $f'_i(p_S(\boldsymbol{\alpha})) \neq 0$ and f'_i and f'_j have disjoint sets of useful parameters. It entails that there exists a minimal support T of f'_j such that $T \subset S$. Moreover, S is a minimal support of f_1, and so $f_k(p_T(\boldsymbol{\alpha})) = 0$ for all $\boldsymbol{\alpha}$ and for all k satisfying $1 \leq k \leq m$. Then, there exists an $\boldsymbol{\alpha}$ such that

$$\prod_{k=1}^{m} R_{P_k}(f_k(p_T(\boldsymbol{\alpha}))) = I \tag{8.10}$$

and

$$\prod_{k=1}^{m'} R_{P'_k}(f'_k(p_T(\boldsymbol{\alpha}))) = R_{P'_j}(f'_j(p_T(\boldsymbol{\alpha})) \neq I \tag{8.11}$$

which contradicts Equation 8.6. Therefore, by contradiction, there do not exist j and $\boldsymbol{\alpha}$ such that $j \neq i$ and $f'_j(p_S(\boldsymbol{\alpha})) \neq 0$. Thus, Equation 8.9 holds, implying that, based on Equation 8.6, we have

$$R_{P_1}(f_1(p_S(\boldsymbol{\alpha}))) = R_{P'_i}(f'_i(p_S(\boldsymbol{\alpha}))) \tag{8.12}$$

for all $\boldsymbol{\alpha}$, and so $P_1 = P'_i$ and

$$f_1(p_S(\boldsymbol{\alpha})) = f'_i(p_S(\boldsymbol{\alpha})) \tag{8.13}$$

for all $\boldsymbol{\alpha}$.

We now prove that $R_{P'_i}(f'_i(\boldsymbol{\alpha}))$ commutes, up to a global phase, with $R_{P'_j}(f'_j(\boldsymbol{\alpha}))$ for all $\boldsymbol{\alpha}$ and for all integer j satisfying $1 \leq j \leq i$. Let j be an integer satisfying $1 \leq j < i$ and such that $R_{P'_j}(f'_j(\boldsymbol{\alpha}))$ anticommutes with $R_{P'_i}(f'_i(\boldsymbol{\alpha}))$ for some $\boldsymbol{\alpha}$. Let T be a minimal support of f'_j and let's assume that there exists $\boldsymbol{\alpha}$ such that $f_1(p_T(\boldsymbol{\alpha})) \neq 0$. Then there would exist a minimal support U of f_1 such that $U \subseteq T$. Based on Equation 8.6, we then have

$$R_{P_1}(f_1(p_U(\boldsymbol{\alpha}))) = R_{P'_j}(f'_j(p_U(\boldsymbol{\alpha}))) \tag{8.14}$$

for all $\boldsymbol{\alpha}$, which entails $P_1 = P'_j$. However, we also have $P_1 = P'_i$, which implies $P'_i = P'_j$. This contradicts our assumption that $R_{P'_j}(f'_j(\boldsymbol{\alpha}))$ anticommutes with $R_{P'_i}(f'_i(\boldsymbol{\alpha}))$ for some $\boldsymbol{\alpha}$. Therefore, by contradiction, we have $f_1(p_T(\boldsymbol{\alpha})) = 0$ for all $\boldsymbol{\alpha}$. It follows that, based on Equation 8.6, we have

$$\prod_{k=1}^{m} R_{P_k}(f_k(p_T(\boldsymbol{\alpha}))) = \prod_{k=2}^{m} R_{P_k}(f_k(p_T(\boldsymbol{\alpha}))) = \prod_{k=1}^{m'} R_{P'_k}(f'_k(p_T(\boldsymbol{\alpha}))) = R_{P'_j}(f'_j(p_T(\boldsymbol{\alpha}))) \tag{8.15}$$

for all $\boldsymbol{\alpha}$, because $T \subseteq \text{par}(f'_j)$ and all the f' functions have disjoint sets of useful parameters. It entails

$$T \subseteq \bigcup_{k=2}^{m} \text{par}(f_k) \tag{8.16}$$

and so

$$T \cap \text{par}(f_1) = \emptyset \tag{8.17}$$

because all the f functions have disjoint sets of useful parameters. It implies that

$$R_{P_1}(f_1(p_{S\cup T}(\boldsymbol{\alpha}))) = R_{P_1}(f_1(p_S(\boldsymbol{\alpha}))) \tag{8.18}$$

for all $\boldsymbol{\alpha}$. Moreover, we have

$$R_{P_k}(f_k(p_{S\cup T}(\boldsymbol{\alpha}))) = R_{P_k}(f_k(p_T(\boldsymbol{\alpha}))) \tag{8.19}$$

for all integer k satisfying $1 < k \leq m$, because $S \subseteq \text{par}(f_1)$ and f_1 and f_k have disjoint sets of useful parameters. Based on Equations 8.15, 8.18 and 8.19, we have

$$\begin{aligned} \prod_{k=1}^{m} R_{P_k}(f_k(p_{S\cup T}(\boldsymbol{\alpha}))) &= R_{P_1}(f_1(p_S(\boldsymbol{\alpha}))) \prod_{k=2}^{m} R_{P_k}(f_k(p_T(\boldsymbol{\alpha}))) \\ &= R_{P_1}(f_1(p_S(\boldsymbol{\alpha})))R_{P'_j}(f'_j(p_T(\boldsymbol{\alpha}))) \end{aligned} \tag{8.20}$$

for all $\boldsymbol{\alpha}$. Furthermore, we have

$$\prod_{k=1}^{m'} R_{P'_k}(f'_k(p_{S\cup T}(\boldsymbol{\alpha}))) = R_{P'_j}(f'_j(p_T(\boldsymbol{\alpha})))R_{P'_i}(f'_i(p_S(\boldsymbol{\alpha}))) \tag{8.21}$$

for all $\boldsymbol{\alpha}$, because $j < i$, $T \subseteq \text{par}(f'_j)$, $S \subseteq \text{par}(f'_i)$ and all f' functions have disjoint sets of useful parameters. According to Equation 8.6, Equations 8.20 and 8.21 should be equal:

$$R_{P_1}(f_1(p_S(\boldsymbol{\alpha})))R_{P'_j}(f'_j(p_T(\boldsymbol{\alpha}))) = R_{P'_j}(f'_j(p_T(\boldsymbol{\alpha})))R_{P'_i}(f'_i(p_S(\boldsymbol{\alpha}))) \tag{8.22}$$

for all $\boldsymbol{\alpha}$. Using Equation 8.12, Equation 8.22 can be rewritten as

$$R_{P'_i}(f'_i(p_S(\boldsymbol{\alpha})))R_{P'_j}(f'_j(p_T(\boldsymbol{\alpha}))) = R_{P'_j}(f'_j(p_T(\boldsymbol{\alpha})))R_{P'_i}(f'_i(p_S(\boldsymbol{\alpha}))) \tag{8.23}$$

for all $\boldsymbol{\alpha}$. Equation 8.23 indicates that $R_{P'_i}(f'_i(\boldsymbol{\alpha}))$ commutes with $R_{P'_j}(f'_j(\boldsymbol{\alpha}))$ for all $\boldsymbol{\alpha}$. This contradicts our assumption that $R_{P'_j}(f'_j(\boldsymbol{\alpha}))$ anticommutes with $R_{P'_i}(f'_i(\boldsymbol{\alpha}))$ for some $\boldsymbol{\alpha}$. Therefore, by contradiction, $R_{P'_i}(f'_i(\boldsymbol{\alpha}))$ commutes, up to a global phase, with $R_{P'_j}(f'_j(\boldsymbol{\alpha}))$ for all $\boldsymbol{\alpha}$ and for all integer j satisfying $1 \leq j \leq i$.

We now prove that

$$R_{P_1}(f_1(\boldsymbol{\alpha})) = R_{P'_i}(f'_i(\boldsymbol{\alpha})) \tag{8.24}$$

for all $\boldsymbol{\alpha}$. To do so, we will prove that

$$\text{par}(f_1) = \text{par}(f'_i) \tag{8.25}$$

by showing that $\text{par}(f'_i) \subseteq \text{par}(f_1)$ and $\text{par}(f_1) \subseteq \text{par}(f'_i)$.

Let's assume that there exists an integer j and an $\boldsymbol{\alpha}$ such that $j \neq i$ and $f'_j(p_{\text{par}(f_1)}(\boldsymbol{\alpha})) \neq 0$. Let T be a minimal support of f'_j such that $T \subseteq \text{par}(f_1)$. Based on Equation 8.6, and because all functions f and f' have disjoint sets of useful parameters, we have

$$R_{P_1}(f_1(p_T(\boldsymbol{\alpha}))) = R_{P'_j}(f'_j(p_T(\boldsymbol{\alpha}))) \tag{8.26}$$

for all $\boldsymbol{\alpha}$, which implies that $P_1 = P'_j = P'_i$ because $R_{P'_j}(f'_j(p_T(\boldsymbol{\alpha}))) \neq I$ for some $\boldsymbol{\alpha}$. Let u be an integer between i and j such that the Pauli rotation $R_{P'_u}(f'_u(\boldsymbol{\alpha}))$ anticommutes with $R_{P'_j}(f'_j(\boldsymbol{\alpha}))$ for some $\boldsymbol{\alpha}$. Notice that u necessarily exists as otherwise the Pauli rotations $R_{P'_i}(f'_i(\boldsymbol{\alpha}))$ and $R_{P'_j}(f'_j(\boldsymbol{\alpha}))$ could be merged into a single Pauli rotation because $P'_i = P'_j$, which would contradict the fact that C' has an optimal number of parametrized rotations. Also, i, u and j must satisfy $i < u < j$ because we proved that $R_{P'_i}(f'_i(\boldsymbol{\alpha}))$ commutes with $R_{P'_k}(f'_k(\boldsymbol{\alpha}))$ for all $\boldsymbol{\alpha}$ and k satisfying $1 \leq k < i$. Let U be a minimal support of f'_u, and let's assume that there exists $\boldsymbol{\alpha}$ such that $f_1(p_U(\boldsymbol{\alpha})) \neq 0$. Then there would exists a minimal support V of f_1 such that $V \subseteq U$. Based on Equation 8.6, we then have

$$R_{P_1}(f_1(p_V(\boldsymbol{\alpha}))) = R_{P'_u}(f'_u(p_V(\boldsymbol{\alpha}))) \tag{8.27}$$

for all $\boldsymbol{\alpha}$. Because there exists $\boldsymbol{\alpha}$ such that $f_1(p_V(\boldsymbol{\alpha})) \neq 0$, Equation 8.27 implies that $P_1 = P'_u$. However, we also have $P_1 = P'_j$ and $P'_j \neq P_u$ because there exists an $\boldsymbol{\alpha}$ such that $R_{P'_j}(f'_j(\boldsymbol{\alpha}))$ anticommutes with $R_{P'_u}(f'_u(\boldsymbol{\alpha}))$. Therefore, by contradiction, we have $f_1(p_U(\boldsymbol{\alpha})) = 0$ for all $\boldsymbol{\alpha}$. It follows that, based on Equation 8.6, we have

$$\prod_{k=1}^{m} R_{P_k}(f_k(p_U(\boldsymbol{\alpha}))) = \prod_{k=2}^{m} R_{P_k}(f_k(p_U(\boldsymbol{\alpha}))) = \prod_{k=1}^{m'} R_{P'_k}(f'_k(p_U(\boldsymbol{\alpha}))) = R_{P'_u}(f'_u(p_U(\boldsymbol{\alpha}))) \tag{8.28}$$

for all $\boldsymbol{\alpha}$, because $U \subseteq \text{par}(f'_u)$ and all the f' functions have disjoint sets of useful parameters. It entails

$$U \subseteq \bigcup_{k=2}^{m} \text{par}(f_k) \tag{8.29}$$

and so

$$U \cap \text{par}(f_1) = \emptyset \tag{8.30}$$

because all the f functions have disjoint sets of useful parameters. It implies that

$$R_{P_1}(f_1(p_{T\cup U}(\boldsymbol{\alpha}))) = R_{P_1}(f_1(p_T(\boldsymbol{\alpha}))) \tag{8.31}$$

for all $\boldsymbol{\alpha}$. Moreover, we have

$$R_{P_k}(f_k(p_{T\cup U}(\boldsymbol{\alpha}))) = R_{P_k}(f_k(p_U(\boldsymbol{\alpha}))) \tag{8.32}$$

for all integer k satisfying $1 < k \leq m$, because $T \subseteq \text{par}(f_1)$ and f_1 and f_k have disjoint sets of useful parameters. Based on Equations 8.28, 8.31 and 8.32, we have

$$\begin{aligned}\prod_{k=1}^{m} R_{P_k}(f_k(p_{T\cup U}(\boldsymbol{\alpha}))) &= R_{P_1}(f_1(p_T(\boldsymbol{\alpha}))) \prod_{k=2}^{m} R_{P_k}(f_k(p_U(\boldsymbol{\alpha}))) \\ &= R_{P_1}(f_1(p_T(\boldsymbol{\alpha})))R_{P'_u}(f'_u(p_U(\boldsymbol{\alpha})))\end{aligned} \tag{8.33}$$

for all $\boldsymbol{\alpha}$. Furthermore, we have

$$\prod_{k=1}^{m'} R_{P'_k}(f'_k(p_{T\cup U}(\boldsymbol{\alpha}))) = R_{P'_u}(f'_u(p_U(\boldsymbol{\alpha})))R_{P'_j}(f'_j(p_T(\boldsymbol{\alpha}))) \tag{8.34}$$

for all $\boldsymbol{\alpha}$, because $u < j$, $U \subseteq \text{par}(f'_u)$, $T \subseteq \text{par}(f'_j)$ and all f' functions have disjoint sets of useful parameters. According to Equation 8.6, Equations 8.33 and 8.34 should be equal:

$$R_{P_1}(f_1(p_T(\boldsymbol{\alpha})))R_{P'_u}(f'_u(p_U(\boldsymbol{\alpha}))) = R_{P'_u}(f'_u(p_U(\boldsymbol{\alpha})))R_{P'_j}(f'_j(p_T(\boldsymbol{\alpha}))) \tag{8.35}$$

for all $\boldsymbol{\alpha}$. Using Equation 8.26, Equation 8.35 can be rewritten as

$$R_{P'_j}(f'_j(p_T(\boldsymbol{\alpha})))R_{P'_u}(f'_u(p_U(\boldsymbol{\alpha}))) = R_{P'_u}(f'_u(p_U(\boldsymbol{\alpha})))R_{P'_j}(f'_j(p_T(\boldsymbol{\alpha}))) \tag{8.36}$$

for all $\boldsymbol{\alpha}$. Equation 8.36 indicates that $R_{P'_u}(f'_u(\boldsymbol{\alpha}))$ commutes with $R_{P'_j}(f'_j(\boldsymbol{\alpha}))$ for all $\boldsymbol{\alpha}$. This contradicts the fact that $R_{P'_u}(f'_u(\boldsymbol{\alpha}))$ anticommutes with $R_{P'_j}(f'_j(\boldsymbol{\alpha}))$ for some $\boldsymbol{\alpha}$. Therefore, by contradiction, there does not exist an integer j and an $\boldsymbol{\alpha}$ such that $j \neq i$ and $f'_j(p_{\text{par}(f_1)}(\boldsymbol{\alpha})) \neq 0$. Thus, based on Equation 8.6,

$$R_{P_1}(f_1(p_{\text{par}(f_1)}(\boldsymbol{\alpha}))) = R_{P'_i}(f'_i(p_{\text{par}(f_1)}(\boldsymbol{\alpha}))) \tag{8.37}$$

for all $\boldsymbol{\alpha}$, which implies $\text{par}(f'_i) \subseteq \text{par}(f_1)$.

Let's assume that there exists an integer j and an $\boldsymbol{\alpha}$ such that $1 < j$ and $f_j(p_{\text{par}(f'_i)}(\boldsymbol{\alpha})) \neq 0$. Let T be a minimal support of f_j such that $T \subseteq \text{par}(f'_i)$. Based on Equation 8.6, and because all functions f and f' have disjoint sets of useful parameters, we have

$$R_{P_j}(f_j(p_T(\boldsymbol{\alpha}))) = R_{P'_i}(f'_i(p_T(\boldsymbol{\alpha}))) \tag{8.38}$$

for all $\boldsymbol{\alpha}$, which implies that $P'_i = P_j = P_1$ because $R_{P_j}(f_j(p_T(\boldsymbol{\alpha}))) \neq I$ for some $\boldsymbol{\alpha}$. Let u be an integer satisfying $1 < u < j$ such that the Pauli rotation $R_{P_u}(f_u(\boldsymbol{\alpha}))$ anticommutes with $R_{P_j}(f_j(\boldsymbol{\alpha}))$ for some $\boldsymbol{\alpha}$. Notice that u necessarily exists as otherwise the Pauli rotations $R_{P_1}(f_1(\boldsymbol{\alpha}))$ and $R_{P_j}(f_j(\boldsymbol{\alpha}))$ could be merged into a single Pauli rotation because $P_i = P_j$, which would contradict the fact that C_o is a circuit produced by Algorithm 10. Let U be a minimal support of f_u, and let's assume that there exists $\boldsymbol{\alpha}$ such that $f'_i(p_U(\boldsymbol{\alpha})) \neq 0$. Then there would exists a minimal support V of f'_i such that $V \subseteq U$. Based on Equation 8.6, we then have

$$R_{P_u}(f_u(p_V(\boldsymbol{\alpha}))) = R_{P'_i}(f'_i(p_V(\boldsymbol{\alpha}))) \tag{8.39}$$

for all $\boldsymbol{\alpha}$. Because there exists $\boldsymbol{\alpha}$ such that $f_i'(p_V(\boldsymbol{\alpha})) \neq 0$, Equation 8.39 implies that $P_i' = P_u$. However, we also have $P_i' = P_j$ and $P_j \neq P_u$ because there exists an $\boldsymbol{\alpha}$ such that $R_{P_j}(f_j(\boldsymbol{\alpha}))$ anticommutes with $R_{P_u}(f_u(\boldsymbol{\alpha}))$. Therefore, by contradiction, we have $f_i'(p_U(\boldsymbol{\alpha})) = 0$ for all $\boldsymbol{\alpha}$. It follows that, based on Equation 8.6, we have

$$\prod_{k=1}^{m'} R_{P_k'}(f_k'(p_U(\boldsymbol{\alpha}))) = \prod_{k=1,k\neq i}^{m'} R_{P_k'}(f_k'(p_U(\boldsymbol{\alpha}))) = \prod_{k=1}^{m} R_{P_k}(f_k(p_U(\boldsymbol{\alpha}))) = R_{P_u}(f_u(p_U(\boldsymbol{\alpha}))) \tag{8.40}$$

for all $\boldsymbol{\alpha}$, because $U \subseteq \mathrm{par}(f_u)$ and all the f functions have disjoint sets of useful parameters. It entails

$$U \subseteq \bigcup_{k=2}^{m} \mathrm{par}(f_k') \tag{8.41}$$

and so

$$U \cap \mathrm{par}(f_i') = \emptyset \tag{8.42}$$

because all the f functions have disjoint sets of useful parameters. It implies that

$$R_{P_i'}(f_i'(p_{T\cup U}(\boldsymbol{\alpha}))) = R_{P_i'}(f_i'(p_T(\boldsymbol{\alpha}))) \tag{8.43}$$

for all $\boldsymbol{\alpha}$. Moreover, we have

$$R_{P_k'}(f_k'(p_{T\cup U}(\boldsymbol{\alpha}))) = R_{P_k'}(f_k'(p_U(\boldsymbol{\alpha}))) \tag{8.44}$$

for all integer k satisfying $1 \leq k \leq m$, $k \neq i$, because $T \subseteq \mathrm{par}(f_i')$ and f_i' and f_k' have disjoint sets of useful parameters. Based on Equations 8.40, 8.43 and 8.44, we have

$$\begin{aligned}\prod_{k=1}^{m'} R_{P_k'}(f_k'(p_{T\cup U}(\boldsymbol{\alpha}))) &= R_{P_i'}(f_i'(p_T(\boldsymbol{\alpha}))) \prod_{k=1,k\neq i}^{m'} R_{P_k'}(f_k'(p_U(\boldsymbol{\alpha}))) \\ &= R_{P_i'}(f_i'(p_T(\boldsymbol{\alpha})))R_{P_u}(f_u(p_U(\boldsymbol{\alpha})))\end{aligned} \tag{8.45}$$

for all $\boldsymbol{\alpha}$ because we proved that $R_{P_i'}(f_i'(\boldsymbol{\alpha}))$ commutes with $R_{P_k'}(f_k'(\boldsymbol{\alpha}))$ for all $\boldsymbol{\alpha}$ and all integer k satisfying $1 \leq k \leq i$. Furthermore, we have

$$\prod_{k=1}^{m} R_{P_k}(f_k(p_{T\cup U}(\boldsymbol{\alpha}))) = R_{P_u}(f_u(p_U(\boldsymbol{\alpha})))R_{P_j}(f_j(p_T(\boldsymbol{\alpha}))) \tag{8.46}$$

for all $\boldsymbol{\alpha}$, because $u < j$, $U \subseteq \mathrm{par}(f_u)$, $T \subseteq \mathrm{par}(f_j)$ and all f functions have disjoint sets of useful parameters. According to Equation 8.6, Equations 8.45 and 8.46 should be equal:

$$R_{P_i'}(f_i'(p_T(\boldsymbol{\alpha})))R_{P_u}(f_u(p_U(\boldsymbol{\alpha}))) = R_{P_u}(f_u(p_U(\boldsymbol{\alpha})))R_{P_j}(f_j(p_T(\boldsymbol{\alpha}))) \tag{8.47}$$

for all $\boldsymbol{\alpha}$. Using Equation 8.38, Equation 8.47 can be rewritten as

$$R_{P_j}(f_j(p_T(\boldsymbol{\alpha})))R_{P_u}(f_u(p_U(\boldsymbol{\alpha}))) = R_{P_u}(f_u(p_U(\boldsymbol{\alpha})))R_{P_j}(f_j(p_T(\boldsymbol{\alpha}))) \tag{8.48}$$

for all $\boldsymbol{\alpha}$. Equation 8.48 indicates that $R_{P_j}(f_j(\boldsymbol{\alpha}))$ commutes with $R_{P_u}(f_u(\boldsymbol{\alpha}))$ for all $\boldsymbol{\alpha}$. This contradicts the fact that $R_{P_u}(f_u(\boldsymbol{\alpha}))$ anticommutes with $R_{P_j}(f_j(\boldsymbol{\alpha}))$ for some $\boldsymbol{\alpha}$. Therefore, by contradiction, there does not exist an integer j and an $\boldsymbol{\alpha}$ such that $1 < j$ and $f_j(p_{\mathrm{par}(f_i')}(\boldsymbol{\alpha})) \neq 0$. Thus, based on Equation 8.6,

$$R_{P_1}(f_1(p_{\mathrm{par}(f_i')}(\boldsymbol{\alpha}))) = R_{P_i'}(f_i'(p_{\mathrm{par}(f_i')}\boldsymbol{\alpha}))) \tag{8.49}$$

for all $\boldsymbol{\alpha}$, which implies $\text{par}(f_1) \subseteq \text{par}(f'_i)$.

We proved that $\text{par}(f'_i) \subseteq \text{par}(f_1)$ and $\text{par}(f_1) \subseteq \text{par}(f'_i)$, which implies $\text{par}(f_1) = \text{par}(f'_i)$. Based on Equation 8.6 and because all f and f' functions have disjoint sets of useful parameters, it follows that

$$R_{P_1}(f_1(\boldsymbol{\alpha})) = R_{P'_i}(f'_i(\boldsymbol{\alpha})) \tag{8.50}$$

for all $\boldsymbol{\alpha}$.

We can now conclude the proof of Lemma 2 by induction. Let k be an integer satisfying $0 < k < m$. Let's assume that there exists an injective function σ satisfying $R_{P_i}(f_i(\boldsymbol{\alpha})) = R_{P'_{\sigma(i)}}(f'_{\sigma(i)}(\boldsymbol{\alpha}))$ for all i satisfying $1 \leq i \leq k$ and such that $R_{P'_{\sigma(i)}}(f'_{\sigma(i)}(\boldsymbol{\alpha}))$ commutes with $R_{P'_j}(f'_j(\boldsymbol{\alpha}))$ for all $\boldsymbol{\alpha}$ and all integer j satisfying $\sigma(i) \leq j \leq i$ and $i \leq j \leq \sigma(i)$. Note that we already proved this in the case where $k = 1$. Let $\tilde{C}_o$ and $\tilde{C}'$ be the parametrized Clifford circuits with non-repeated parameters obtained from C_o and C', respectively, by removing the parametrized rotation gates associated with the Pauli rotations $R_{P_i}(f_i(\boldsymbol{\alpha}))$ and $R_{P'_{\sigma(i)}}(f'_{\sigma(i)}(\boldsymbol{\alpha}))$ for all i satisfying $1 \leq i \leq k$. The parametrized Clifford circuits $\tilde{C}_o$ and $\tilde{C}'$ are equivalent because $R_{P_i}(f_i(\boldsymbol{\alpha})) = R_{P'_{\sigma(i)}}(f'_{\sigma(i)}(\boldsymbol{\alpha}))$ for all i satisfying $1 \leq i \leq k$, and $P'_{\sigma(i)}$ commutes with P'_j for all j satisfying $\sigma(i) \leq j \leq i$ and $i \leq j \leq \sigma(i)$. Also, because C_o is produced by Algorithm 10, there does not exist two integers i, j satisfying $k < i < j \leq m$ and $P_i = P_j$, and such that $R_{P_j}(f_j(\boldsymbol{\alpha}))$ commutes with $R_{P_\ell}(f(_\ell(\boldsymbol{\alpha}))$ for all integer ℓ satisfying $i \leq \ell \leq j$. Therefore, the parametrized Clifford circuit with non-repeated parameters produced by Algorithm 10 when $\tilde{C}_o$ is given as input is identical to $\tilde{C}_o$. Moreover, and $\tilde{C}'$ has an optimal number of parametrized rotations because C' has an optimal number of parametrized rotations. Thus, the circuits $\tilde{C}_o$ and $\tilde{C}'$ are satisfying the conditions of Lemma 2. Then, as proven above, $R_{P_{k+1}}(f_{k+1}(\boldsymbol{\alpha})) = R_{P'_i}(f'_i(\boldsymbol{\alpha}))$ for all $\boldsymbol{\alpha}$ and some integer i where $R_{P'_i}(f'_i(\boldsymbol{\alpha}))$ belongs to the sequence of Pauli rotations associated with $\tilde{C}'$, and $R_{P'_i}(f'_i(\boldsymbol{\alpha}))$ commutes, up to a global phase, with $R_{P'_j}(f'_j(\boldsymbol{\alpha}))$ for all j satisfying $i \leq j \leq k+1$ and $k+1 \leq j \leq k+1$. Hence, by induction, $m = m'$ and there exists a permutation σ satisfying $R_{P_i}(f_i(\boldsymbol{\alpha})) = R_{P'_{\sigma(i)}}(f'_{\sigma(i)}(\boldsymbol{\alpha}))$ for all i and such that $R_{P'_{\sigma(i)}}(f'_{\sigma(i)}(\boldsymbol{\alpha}))$ commutes, up to a global phase, with $R_{P'_j}(f'_j(\boldsymbol{\alpha}))$ for all $\boldsymbol{\alpha}$ and all integer j satisfying $\sigma(i) \leq j \leq i$ and $i \leq j \leq \sigma(i)$. □

A direct consequence of Lemma 2 is that the relations between the parameters of two equivalent parametrized Clifford circuits with non-repeated parameters are corresponding to additive relations. In Reference [138], it was proven that parameter transformations restricted to analytic functions are actually equal to affine functions. Lemma 2 extends this result for the general case, thereby showing that affine functions are truly all we need to describe the parameter transformations.

Theorem 12 is not exclusive to Algorithm 10; the result also holds for any algorithm which finds all the parametrized rotations that can be merged together. Among these algorithms, Algorithm 10 is the one having the best complexity. As discussed in Chapter 7, the algorithm of Reference [110] has a complexity of $\mathcal{O}(nM + nm^2)$ where n is the number of qubits, M is the number of gates and m is the number of parametrized rotations. This is worse than the $\mathcal{O}(nM + nhm)$ complexity of Algorithm 10, where h satisfies $h < m$ and is the minimal number of internal Hadamard gates. In Reference [109], another approach is proposed with a complexity of $\mathcal{O}(N^3)$, where N is the number of spiders in the ZX-diagram associated with the initial circuit. This complexity is also worse than the one of Algorithm 10 because N satisfies $h < m \leq N$ and $n \leq N$.

8.3 Optimality in the number of Hadamard gates and internal Hadamard gates

In this section, we present similar results for the number of Hadamard gates and internal Hadamard gates. The number of internal Hadamard gates in a parametrized quantum circuit corresponds to the number of Hadamard between the first and last parametrized rotation gate of the circuit. In Chapter 6, an algorithm is presented for implementing the sequence of Pauli rotations and the following Clifford operator of Equation 8.2 with a minimal number of Hadamard gates and internal Hadamard gates over the $\{\text{CNOT}, H, S, X, R_Z\}$ gate set. In this section, we will base our results on this algorithm and we will also consider that the Clifford gates in parametrized Clifford circuits are from the $\{\text{CNOT}, H, S, X\}$ gate set. We prove the following theorem, which states the optimality of the algorithm presented in Chapter 6 for reducing the number of Hadamard and internal Hadamard gates in a parametrized Clifford circuit with non-repeated parameters.

Theorem 13. *Let C_o be the parametrized Clifford circuit with non-repeated parameters produced by the algorithm presented in Chapter 6 when a parametrized Clifford circuit with non-repeated parameters is given as input. Then, he number of Hadamard gates and the number of internal Hadamard gates in C_o is optimal: there does not exist a parametrized Clifford circuit with non-repeated parameters equivalent to C_o which contains a lower number of Hadamard gates or a lower number of internal Hadamard gates.*

We will rely on the following lemma to prove Theorem 13, which states that applying Algorithm 10 on a parametrized Clifford circuit do not change the rank of its associated commutativity matrix.

Lemma 3. *Let C be a parametrized Clifford circuits with non-repeated parameters and let C_o be the parametrized Clifford circuit with non-repeated parameters produced by Algorithm 10 when C is given as input. Let A and A_o be the commutativity matrices associated with C and C_o respectively. Then* $\text{rank}(A) = \text{rank}(A_o)$.

Proof. Let $R_{P_1}(f_1(\boldsymbol{\alpha})), \ldots, R_{P_m}(f_m(\boldsymbol{\alpha}))$ be a sequence of Pauli rotations associated with C. Let i be an integer satisfying $1 \leq i < m$ and such that P_i commutes with P_{i+1}. Let $\tilde{A}$ be the commutativity matrix associated with the sequence of Pauli rotations where the positions of the ith and $(i+1)$th Pauli rotations have been swapped:

$$R_{P_1}(f_1(\boldsymbol{\alpha})), \ldots, R_{P_{i-1}}(f_{i-1}(\boldsymbol{\alpha})), R_{P_{i+1}}(f_{i+1}(\boldsymbol{\alpha})), R_{P_i}(f_i(\boldsymbol{\alpha})), R_{P_{i+2}}(f_{i+2}(\boldsymbol{\alpha})), \ldots, R_{P_m}(f_m(\boldsymbol{\alpha})).$$

Because P_i commutes with P_{i+1}, by definition we have $A_{i,i+1} = \tilde{A}_{i,i+1} = 0$. It follows that $A_{j,i} = \tilde{A}_{j,i+1}$, $A_{j,i+1} = \tilde{A}_{j,i}$ and $A_{i,j} = \tilde{A}_{i+1,j}$, $A_{i+1,j} = \tilde{A}_{i,j}$ for all j. I.e. the matrix $\tilde{A}$ can be obtained from the matrix A by swapping its ith and $(i+1)$th columns and its ith and $(i+1)$th rows. Performing these operations on the matrix A do not change its rank, therefore $\text{rank}(A) = \text{rank}(\tilde{A})$.

Let i be an integer satisfying $1 \leq i < m$ and such that $P_i = P_{i+1}$. Let $\tilde{A}$ be the commutativity matrix associated with the sequence of Pauli rotations where the ith and $(i+1)$th Pauli rotations have been merged into a single Pauli rotation:

$$R_{P_1}(f_1(\boldsymbol{\alpha})), \ldots, R_{P_i}(f_i(\boldsymbol{\alpha}) + f_{i+1}(\boldsymbol{\alpha})), R_{P_{i+2}}(f_{i+2}(\boldsymbol{\alpha})), \ldots, R_{P_m}(f_m(\boldsymbol{\alpha})).$$

Because $P_i = P_{i+1}$, by definition we have $A_{j,i} = A_{j,i+1} = \tilde{A}_{j,i}$ and $A_{i,j} = A_{i+1,j} = \tilde{A}_{i,j}$ for all j. I.e. the ith column of A is equal to the $(i+1)$th column of A, the ith row of A is equal to the

$(i+1)$th row of A, and the matrix $\tilde{A}$ can be obtained from the matrix A by removing its $(i+1)$th column and its $(i+1)$th row. Performing these operations on the matrix A do not change its rank, therefore $\text{rank}(A) = \text{rank}(\tilde{A})$.

As stated by Lemma 2, the sequence of Pauli rotations associated with C_o can be obtained from the sequence of Pauli rotations associated with C by repeating any of the two following operations a finite number of times:

- Swap the positions of two adjacent commuting Pauli rotations.
- Merge two adjacent Pauli rotations $R_{P_i}(f_i(\boldsymbol{\alpha}))$ and $R_{P_{i+1}}(f_{i+1}(\boldsymbol{\alpha}))$ satisfying $P_i = P_{i+1}$ into a single Pauli rotation $R_{P_i}(f_i(\boldsymbol{\alpha}) + f_{i+1}(\boldsymbol{\alpha}))$.

We proved that these two operations do not change the rank of the associated commutativity matrix. Thus, $\text{rank}(A) = \text{rank}(A_o)$. □

Based on Lemma 3, we can prove the following lemma, which states that the commutativity matrices and the extended commutativity matrices associated with the parametrized Clifford circuits with non-repeated parameters produced by Algorithm 10 all have the same rank.

Lemma 4. *Let C and C' be equivalent parametrized Clifford circuits with non-repeated parameters. Let A and A' be the commutativity matrices associated with C and C' respectively. And let M and M' be the extended commutativity matrices associated with C and C' respectively. Then* $\text{rank}(A) = \text{rank}(A')$ *and* $\text{rank}(M) = \text{rank}(M')$.

Proof. Let C_o and C'_o be the parametrized Clifford circuits with non-repeated parameters produced by Algorithm 10 when C and C' are given as input. And let A_o and A'_o be the commutativity matrices associated with C_o and C'_o respectively. Based on Lemma 3 we have $\text{rank}(A_o) = \text{rank}(A)$ and $\text{rank}(A'_o) = \text{rank}(A')$. Moreover, Lemma 2 states that C_o and C'_o have the same number of parametrized rotations, and that the sequence of Pauli rotations associated with C_o can be obtained from the sequence of Pauli rotations associated with C'_o by repeatedly swapping the position of two adjacent commuting Pauli rotations. As shown in the proof Lemma 3, swapping the position of two adjacent commuting Pauli rotations do not change the rank of the associated commutativity matrix. It follows that $\text{rank}(A_o) = \text{rank}(A'_o)$, which implies that $\text{rank}(A) = \text{rank}(A')$.

Let $\tilde{C}$ and $\tilde{C}'$ be circuits obtained from C and C' by inserting a parametrized R_Z gate at the beginning and the end of the circuit on each qubit, such that $\tilde{C}$ and $\tilde{C}'$ are equivalent parametrized Clifford circuits with non-repeated parameters. As proven above, the commutativity matrices associated with $\tilde{C}$ and $\tilde{C}'$ have the same rank because $\tilde{C}$ and $\tilde{C}'$ are equivalent. Moreover, the commutativity matrices of $\tilde{C}$ and $\tilde{C}'$ are the extended commutativity matrices of C and C' respectively, denoted M and M'. Thus, $\text{rank}(M) = \text{rank}(M')$. □

The proof of Theorem 13 can now be formulated by relying on Lemma 4.

Proof of Theorem 13. Let A_o and M_o be the commutativity matrix and the extended commutativity matrix associated with C_o. The algorithm presented in Chapter 6 produces a circuit with a number of internal Hadamard gates equal to $\text{rank}(A_o)$ and a number of Hadamard gates equal to $\text{rank}(M_o)$. Lemma 4 states that any parametrized Clifford circuits with non-repeated parameters equivalent to C_o with an associated commutativity matrix A and an associated extended commutativity matrix M satisfies $\text{rank}(A) = \text{rank}(A_o)$ and $\text{rank}(M) = \text{rank}(M_o)$. Moreover, in a quantum circuit composed of gates from the set $\{\text{CNOT}, S, H, X, R_Z\}$, the number of internal Hadamard gates and the number of Hadamard gates cannot be lower than the rank of its

associated commutativity matrix and the rank of its associated extended commutativity matrix respectively. Therefore, there does not exist a parametrized Clifford circuit with non-repeated parameters equivalent to C_o which contains a number of internal Hadamard lower than rank(A_o) or a number of Hadamard gates lower than rank(M_o). □

Chapter 9

T-Count Optimization

One of the main tasks of a quantum compiler is to minimize the resources needed to execute a given quantum algorithm. This step is of considerable importance in the compilation stack as it makes quantum computation more practical and efficient. To complete this task effectively and decide which optimization to perform, we must first identify the most expensive operations hindering our way towards fast and functional quantum computation. In this regard the T gate is often targeted as it cannot be trivially implemented, for instance via transversal operations, in a fault-tolerant way in most quantum error correcting codes as opposed to Clifford gates. It implies that implementing a T gate is generally much more costly than performing a Clifford operation [139–141]. Also, numerous quantum error correcting codes can be used to perform universal fault-tolerant quantum computation with the Clifford+T gate set. In such a setup, the depth of a quantum circuit, and so the time required to execute it, is generally directly linked to the T-depth of the circuit [142]. For this reason, much work has been put into the minimization of the T-depth in Clifford+T circuits [104, 105, 143–146]. But if the number of qubits at disposal is limited, then the depth of the circuit also depends on the number of T gates within it. In addition, the depth of a circuit dictates the minimum number of physical qubits needed to encode the logical qubits utilized to perform the fault-tolerant computation. The coherence time of the logical qubits must be greater than the time required to execute the whole circuit, and so the required amount of physical qubits per logical qubits increases as the depth of the circuit increases. We can discern a feedback loop here: if the depth of the circuit is diminished then less physical qubits are required to encode the logical qubits, which frees up qubits that can be used to further lower the depth of the circuit. The optimization of the T-count can intervene at multiple stages of this process. Firstly, a lower T-count can induce a lower T-depth and so a lower circuit depth [105]. Also, the number of qubits required to implement the T gates can depend on the T-count. This is for example the case when the T gates are implemented via magic state distillation [38]. A lower number of T gates can thus lower the number of physical qubits required to implement the circuit.

The optimization of the T-count also have important applications outside of fault-tolerant quantum computation. Numerous quantum compilers designed for NISQ devices are incorporating a step consisting in reducing the number of T gates [132–134]. It has been demonstrated by these compilers that reducing the T-count can lead to shorter circuits and can help with the minimization of other gates such as the CNOT gate.

Besides compilation, T-count minimization also plays an important role in quantum circuit simulation. As stated by the Gottesman-Knill theorem [18], circuits composed of Clifford gates and Pauli measurements can be efficiently simulated by a classical computer. Extending the

gate set by adding the T gate allows the simulation of universal quantum circuits at the cost of a significant increase in computational time as no algorithm is currently known to efficiently simulate these circuits. That is why many simulation techniques have a runtime that scales exponentially with respect to the number of T gates [115–119]. Minimizing the number of T gates is then essential to exploit the full potential of these simulators and to push back the frontier of non-simulable quantum circuits.

State of the art and contributions. Any unitary gate can be implemented by the Clifford+T gate set up to an arbitrary precision. Therefore, a compilation problem of primordial importance is to find an approximation, over the Clifford+T gate set and that uses a little amount of T gates, of a given unitary operator to an accuracy within $\epsilon > 0$. A fundamental solution to this problem, and which can be applied to any finite universal gate set, was given by the Solovay-Kitaev theorem [19, 20]. Other approaches designed for the Clifford+T gate set were then developed to obtain better approximations [147–149]. Further improvements have then been made by introducing measurements [150] and by using a probabilistic mixture of unitaries [151, 152]. Recently, it has been shown that these two methods can be combined with a novel approach to achieve better results [153]. Another important synthesis problem concerns the set of unitary gates which can be exactly implemented over the Clifford+T gate set. Given one of these unitary gates, the problem then consists in finding an exact Clifford+T implementation of it using a minimal number of T gates. For this problem, an optimal and efficient algorithm is known for the case of single-qubit unitaries [154].

Once a Clifford+T implementation of a unitary gate has been found, whether through approximate or exact synthesis, some quantum circuit optimization methods can then be applied to reduce the number of T gates in the circuit. The algorithms developed for this purpose and achieving the best T-count reductions are foremostly designed for the restricted class of $\{\text{CNOT}, S, T\}$ quantum circuits. The problem of T-count optimization for this class of circuits has been well defined by showing its equivalence with the problem of decoding Reed-Muller codes [106]. In particular, it was demonstrated that the codewords of the punctured Reed-Muller code of length $2^n - 1$ and order $n-4$ are generating the complete set of identities that can be used to optimize the number of T gates in $\{\text{CNOT}, S, T\}$ circuits. Reducing the number of T gates can then be done by finding relevant identities in this large set. For example it has been shown that a particular subset of identities, called spider nest identities, can be efficiently exploited to reduce the number of T gates [111–113, 155]. An effective way to find relevant identities that can be applied to reduce the number of T gates was given by the `TODD` algorithm [108]. However, an important drawback of the `TODD` algorithm is its complexity of $\mathcal{O}(n^3m^5)$ where n is the number of qubits and m is the number of T gates in the initial circuit, which makes it impractical for circuits of large size. In Section 9.2, we show how the complexity of the `TODD` algorithm can be reduced to $\mathcal{O}(n^4m^3)$. In addition, we propose some modifications to the `TODD` algorithm which are resulting in a significantly improved reduction in the number of T gates. In the same section we propose another algorithm which has an even lower complexity of $\mathcal{O}(n^2m^3)$ and that achieves better results than the original `TODD` algorithm on most quantum circuits evaluated. We also prove that the algorithms presented in Section 9.2 are producing Hadamard-free circuits in which the T-count is upper bounded by $(n^2+n)/2+1$ where n is the number of qubits. Benchmarks are provided in Section 9.3 to evaluate the performances, in terms of T-count and execution time, of our algorithms on a library of reversible logic circuits and on large-scale quantum circuits. In Section 9.4, we extend our results for minimizing the number of $R_Z(\pi/2^d)$ gates, where d is a non-negative integer. Finally, in Section 9.5, we demonstrate an upper bound for the number of $R_Z(\pi/2^d)$ gates in a Clifford+$\{R_Z(\pi/2^d), R_Z(2\pi/2^d)\}$ circuit. For Clifford+T circuits we obtain

Algorithm 12: Grouping of Pauli rotations

Input: A sequence $R_{P_1}, \ldots, R_{P_m}$ of Pauli rotations.
Output: Equivalent sequence as layers of mutually commuting Pauli rotations.

```
L ← list of empty sets
for i ← 1 to m do
    j ← max({j | R_{P_i} anticommutes with one element in L_j} ∪ {0})
    L_{j+1} ← L_{j+1} ∪ {R_{P_i}}
end
return L
```

an upper bound of $(n+1)(n+2h)/2+1$ for the number of T gates, which can be satisfied in polynomial time and without any ancillary qubit, and where h is the number of internal Hadamard gates in the circuit.

9.1 Problem statement

9.1.1 T-count optimization in Hadamard-free circuits

Multiple algorithms to reduce the number of T gates are foremostly designed for circuits composed of $\{\text{CNOT}, S, T\}$ gates. In order to make use of these algorithms for Clifford+T circuits, it is necessary to circumvent the Hadamard gates in the input circuit since these algorithms cannot be directly executed on them. This can be done using one the two following methods. The first method consists in dividing the circuit into Hadamard-free subcircuits and Clifford subcircuits containing Hadamard gates. The number of T gates in the Hadamard-free subcircuits can then be optimized using these algorithms. There exists multiple strategies that can be employed to create such a partition of the circuit. It is generally preferred to regroup the T gates in the least number of Hadamard-free subcircuits as possible to take advantage of the fact that the number of T gates in an Hadamard-free circuit can be upper bounded by $\mathcal{O}(n^2)$, where n is the number of qubits. One approach to partition the circuit is to describe the operation performed by the circuit by a sequence of Pauli rotations and a final Clifford operator, as in Equation 1.81, and to then reorder the Pauli rotations in the sequence by forming groups of mutually commuting Pauli rotations. This can for example be done by using the procedure whose pseudo-code is given in Algorithm 12. This algorithm has a complexity of $\mathcal{O}(nm^2)$ since checking whether or not two Pauli rotations commute takes $\mathcal{O}(n)$ operations and such commutativity checks are done at most m times at each iteration of the loop. This way of grouping the Pauli rotations is not new, an equivalent algorithm (but which has a worst-case complexity of $\mathcal{O}(nm^3)$) was given in Reference [24]. By using the layers of Pauli rotations produced by Algorithm 12, Equation 1.81 can then be rewritten as follows:

$$U = e^{i\phi} C \left(\prod_{i=1}^{|L|} \prod_{R_P \in L_i} R_P(\pi/4) \right) \tag{9.1}$$

where $|L|$ denotes the number of layers in L.

A Pauli operator P and a Pauli rotation $R_P(\theta)$ are diagonal if and only if P is a tensor product of the matrices I and Z, up to a multiplicative factor of ± 1. Such a Pauli rotation can be implemented without using any Hadamard gate. Because the Pauli rotations in each layer

L_i of Equation 9.1 are mutually commuting, we can find a Clifford operator $C_i \in \mathcal{C}_n$ such that $C_i^\dagger R_P(\pi/4) C_i$ is diagonal for all $R_P \in L_i$. We say that C_i simultaneously diagonalize the Pauli rotations in L_i. A circuit implementing the Clifford operator C_i can be found with a complexity of $\mathcal{O}(n^2 m)$, where n is the number of qubits and m is the number of Pauli rotations in L_i [156]. By performing a simultaneous diagonalization for each layer of Pauli rotations, Equation 9.1 can then be rewritten as follows:

$$\begin{aligned} U &= e^{i\phi} C \left(\prod_{i=|L|}^{1} C_i C_i^\dagger \left(\prod_{R_P \in L_i} R_P(\pi/4) \right) C_i C_i^\dagger \right) \\ &= e^{i\phi} C \left(\prod_{i=|L|}^{1} C_i \left(\prod_{R_P \in L_i} R_{C_i^\dagger P C_i}(\pi/4) \right) C_i^\dagger \right) \end{aligned} \tag{9.2}$$

where $C_i \in \mathcal{C}_n$ and $C_i^\dagger P C_i$ is diagonal. Because $C_i^\dagger P C_i$ is diagonal, the associated Pauli rotations $R_{C_i^\dagger P C_i}(\pi/4)$ can be implemented using exclusively CNOT, S and T gates. The set of Pauli rotations $R_{C_i^\dagger P C_i}(\pi/4)$ for a fixed i and where $R_P \in L_i$ can then be implemented into the same Hadamard-free subcircuit in which the number of T gates can be optimized.

The second method to circumvent the Hadamard gates in the circuit relies on a measurement-based gadget which can substitute a Hadamard gate [121]. This gadget, presented in Figure 1.2, involves an ancilla qubit, a CZ gate and a measurement. If all the Hadamard gates in the circuits are gadgetized, then the circuit is Hadamard-free and the number of T gates can be optimized using algorithms specifically designed for Hadamard-free circuits. Only internal Hadamard gates, which are the Hadamard gates comprised between the first and the last T gates of the circuit, are required to be gadgetized. Indeed, if only the internal Hadamard gates are gadgetized then the circuit can be partitioned into an initial and final Clifford circuit and a Hadamard-free circuit in between containing all the T gates. The main drawback of this method is that one additional qubit must be used for each internal Hadamard gate that is gadgetized. This motivates the minimization of internal Hadamard gates. To do so, we will use the algorithm presented in Chapter 6 which performs the synthesis of the sequence of Pauli rotations of Equation 1.81 with a minimal number of internal Hadamard gates.

The classically controlled X gate of Figure 1.2, can be commuted through the subsequent R_Z gates of the circuit by relying on the following equality:

$$R_Z(\theta) X = X R_Z(-\theta) = X R_Z(-2\theta) R_Z(\theta) \tag{9.3}$$

In a Clifford+T circuit, the angle θ is a multiple of $\pi/4$, and so -2θ is a multiple of $\pi/2$, which results in a rotation implementable using only Clifford gates. The classically controlled X gate is only commuted through the Pauli rotations succeeding the associated Hadamard gate, and not the Pauli rotations preceding it. Therefore, the optimized circuit in which all Hadamard gates have been gadgetized can be implemented with a measurement depth equal to $|L| - 1$, where $|L|$ corresponds to the number of layers as in Equation 9.2. An example of classically controlled Clifford gates resulting from the gadgetization of a Hadamard gate is provided in Figure 9.1.

9.1.2 Weighted polynomial and signature tensor

We now describe the established formalism for the optimization of the number of T gates in $\{\text{CNOT}, S, T\}$ circuits. Let C be a $\{\text{CNOT}, S, T\}$ circuit operating over n qubits. It has been

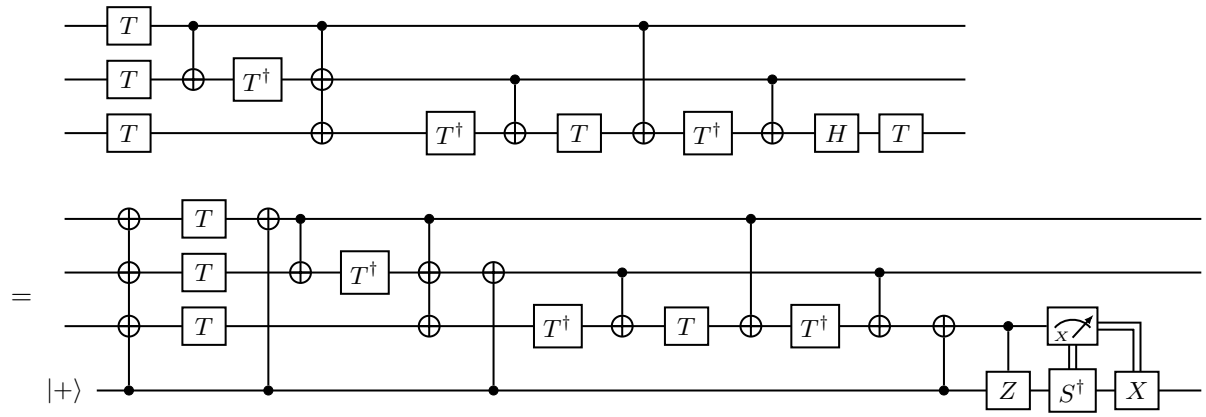


Figure 9.1: Example of classically controlled Clifford gates resulting from the gadgetization of a Hadamard gate and allowing the optimization of the number of T gates.

demonstrated [143] that the action of C on a basis state has the form

$$|\boldsymbol{x}\rangle \mapsto U_f|g(\boldsymbol{x})\rangle \tag{9.4}$$

where $g : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is a linear reversible Boolean function which can be implemented using only CNOT gates [17], and

$$U_f = \sum_{\boldsymbol{x}\in\mathbb{Z}_2^n} \omega^{f(\boldsymbol{x})}|\boldsymbol{x}\rangle\langle\boldsymbol{x}| \tag{9.5}$$

where $\omega = e^{i\pi/4}$ and $f : \mathbb{Z}_2^n \to \mathbb{Z}_8$ is a multilinear polynomial of degree 3 such that

$$f(\boldsymbol{x}) = \sum_{\alpha}^{n} l_\alpha x_\alpha - 2\sum_{\alpha<\beta}^{n} q_{\alpha,\beta}x_\alpha x_\beta + 4\sum_{\alpha<\beta<\gamma}^{n} c_{\alpha,\beta,\gamma}x_\alpha x_\beta x_\gamma \pmod 8 \tag{9.6}$$

where $l_\alpha \in \mathbb{Z}_8$, $q_{\alpha,\beta} \in \mathbb{Z}_4$ and $c_{\alpha,\beta,\gamma} \in \mathbb{Z}_2$. In the same way as in Reference [157], we will refer to the function f as a weighted polynomial due to the fact that each monomial of order m has a coefficient weighted by 2^{m-1}. It has been shown in Reference [157] that U_f belongs to the diagonal subgroup of the third level of the Clifford hierarchy [158]. We will use $\mathcal{D}_3$ to denote this group and $\mathcal{D}_3^C$ to refer to the unitaries of $\mathcal{D}_3$ which are implementable using only CCZ gates.

The parities of the coefficients for a weighted polynomial f can be described by the signature tensor $\mathcal{A}^{(U_f)} \in \mathbb{Z}_2^{(n,n,n)}$ [108], such that $\mathcal{A}^{(U_f)}$ is a symmetric tensor of order 3 satisfying

$$\begin{aligned} \mathcal{A}_{\alpha,\alpha,\alpha} &\equiv l_\alpha && \pmod 2 \\ \mathcal{A}_{\sigma(\alpha,\beta,\beta)} &\equiv \mathcal{A}_{\sigma(\alpha,\alpha,\beta)} = q_{\alpha,\beta} && \pmod 2 \\ \mathcal{A}_{\sigma(\alpha,\beta,\gamma)} &\equiv c_{\alpha,\beta,\gamma} && \pmod 2 \end{aligned} \tag{9.7}$$

where α, β, γ are satisfying $0 \le \alpha < \beta < \gamma < n$ and σ denotes all permutations of the indices. For convenience, we will drop the superscript (U_f) from $\mathcal{A}$ when it is clear from the context that $\mathcal{A}$ is associated with U_f. It has been proven in Reference [157] that U_{2f} can be implemented using only Clifford gates for any weighted polynomial f. It implies that two unitaries U_f and $U_{f'}$, where f and f' are weighted polynomials, are Clifford equivalent if the coefficients of f and f' all have the same parity, i.e. if $\mathcal{A}^{(U_f)}$ and $\mathcal{A}^{(U_{f'})}$ are equal.

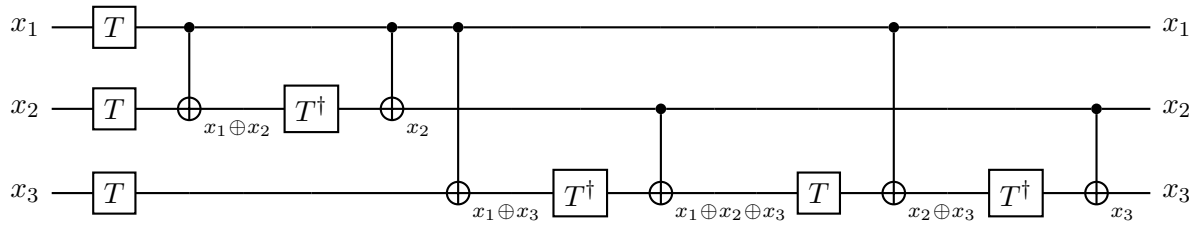


Figure 9.2: An implementation of the U_f gate with weighted polynomial $f(\boldsymbol{x}) = 4x_1x_2x_3$, which corresponds to the CCZ gate.

9.1.3 Phase polynomial and parity table

An implementation of the U_f gate for a weighted polynomial f can be described by a phase polynomial via the circuit-polynomial correspondence [159, 160]. A phase polynomial p is a linear combination of linear Boolean functions:

$$p(\boldsymbol{x}) = \sum_{i=1}^{m} a_i(y_1^{(i)}x_1 \oplus \ldots \oplus y_n^{(i)}x_n) \pmod 8 \tag{9.8}$$

where $\boldsymbol{y}^{(i)} \in \mathbb{Z}_2^n \setminus \{\boldsymbol{0}\}$, $\boldsymbol{a} \in \mathbb{Z}_8^n$ an $m \geq 0$. We will refer to the Boolean vectors $\boldsymbol{y}^{(i)}$ as the parities of the phase polynomial p and to $\boldsymbol{a}$ as the weights of p. The parities of a phase polynomial can be described by a Boolean matrix, called parity table and denoted P, of size $n \times m$ where n is the number of qubits and m is the number of parities weighted by a non-zero a_i. For every weighted polynomial f we can find a phase polynomial p with weights $\boldsymbol{a}$ such that $p(\boldsymbol{x}) = f(\boldsymbol{x})$ for all $\boldsymbol{x}$. Such phase polynomial can then be used to implement U_f via a phase polynomial synthesis algorithm which will result in a circuit containing $|\boldsymbol{a} \pmod 2|$ T gates. As we are focusing on minimizing the number of T gates, we will represent a parity $\boldsymbol{y}^{(i)}$ in the parity table P if and only if its weight is satisfying $a_i \equiv 1 \pmod 2$. The number of columns of P is then equal to the number of T gates required to implement the phase polynomial p. For example, the weighted polynomial associated with the CCZ gate is $f(x_1, x_2, x_3) = 4x_1x_2x_3$. Its symmetric tensor $\mathcal{A} \in \mathbb{Z}_2^{(3,3,3)}$ satisfies:

$$\mathcal{A}_{\alpha,\beta,\gamma} = \begin{cases} 1 & \text{if } \alpha \neq \beta \neq \gamma, \\ 0 & \text{otherwise.} \end{cases} \tag{9.9}$$

One possible phase polynomial that can be use to implement this weighted polynomial is $p(x_1, x_2, x_3) = x_1 + x_2 + x_3 + 7(x_1 \oplus x_2) + 7(x_1 \oplus x_3) + 7(x_2 \oplus x_3) + (x_1 \oplus x_2 \oplus x_3)$. The parity table P and the weights $\boldsymbol{a}$ associated with this phase polynomial are

$$P = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}, \quad \boldsymbol{a} = \begin{pmatrix} 1 & 1 & 1 & 7 & 7 & 7 & 1 \end{pmatrix}^T.$$

We can verify that $f(\boldsymbol{x}) = p(\boldsymbol{x})$ for all $\boldsymbol{x}$ by using the identity $2xy \equiv x+y+7(x \oplus y) \pmod 8$ [144]. The synthesis of a phase polynomial p with weights satisfying $a_i \in \mathbb{Z}_8$ for all i can be realized using CNOT gates and one R_Z gate with an angle of $a_i\pi/4$ for each parity $\boldsymbol{y}^{(i)}$ in p. A circuit implementing the phase polynomial of this example with the CCZ gate is represented in Figure 9.2.

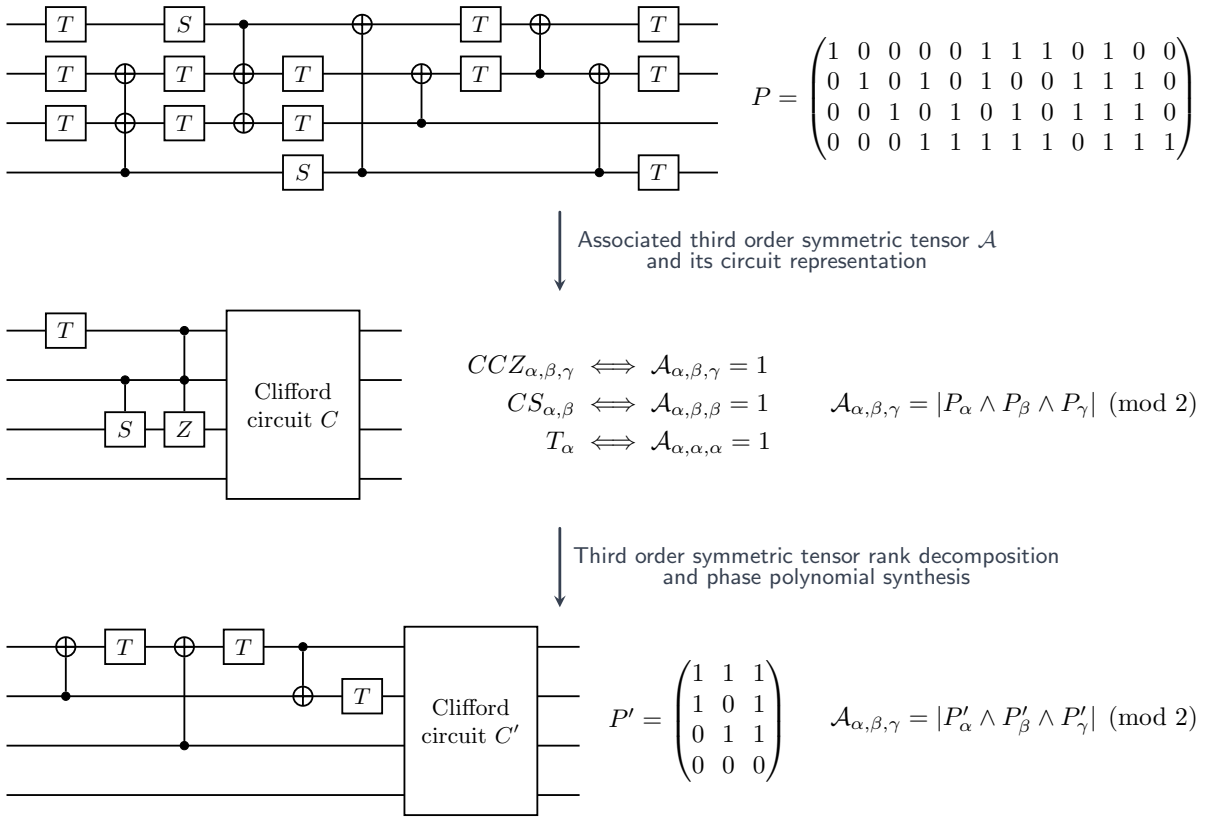


Figure 9.3: Overview of the process for the optimization of the number of T gates in Hadamard-free circuits. We first compute the parity table P associated with the phase polynomial implemented by the initial circuit. Then, we compute the weighted polynomial associated with P, which can be represented by the third order symmetric tensor $\mathcal{A}$ or by a $\{T, CS, CCZ\}$ circuit. We then solve the third order symmetric tensor rank decomposition problem to find another parity table P' associated with $\mathcal{A}$ but which contains a minimal number of columns. Finally, we perform the synthesis of the phase polynomial represented by P' along with the final Clifford operator to obtain the optimized circuit.

The weighted polynomial associated with a phase polynomial composed of a single parity $p(\boldsymbol{x}) = a(x_1 \oplus \ldots \oplus x_n)$, where $a \in \mathbb{Z}_8$, can be computed using the following equality:

$$a\,(x_1 \oplus \ldots \oplus x_n) = a\left(\sum_{\alpha}^{n} x_\alpha - \sum_{\alpha<\beta}^{n} 2x_\alpha x_\beta + \sum_{\alpha<\beta<\gamma}^{n} 4x_\alpha x_\beta x_\gamma\right) \pmod 8. \tag{9.10}$$

We provide a proof of Equation 9.10, and its more general form, in Section 9.4.1. The coefficients of the weighted polynomial represented in the right-hand side of this equation are $l_\alpha = a \pmod 8$, $q_{\alpha,\beta} = a \pmod 4$ and $c_{\alpha,\beta,\gamma} = a \pmod 2$. Equation 9.10 can then be used for each parity of a

given phase polynomial to compute its associated weighted polynomial:

$$\begin{aligned} p(\boldsymbol{x}) &= \sum_{i=1}^{m} a_i (y_1^{(i)} x_1 \oplus \ldots \oplus y_n^{(i)} x_n) \pmod 8 \\ &= \sum_{i=1}^{m} a_i \left(\sum_{\alpha}^{n} y_\alpha^{(i)} x_\alpha - \sum_{\alpha<\beta}^{n} 2 y_\alpha^{(i)} y_\beta^{(i)} x_\alpha x_\beta + \sum_{\alpha<\beta<\gamma}^{n} 4 y_\alpha^{(i)} y_\beta^{(i)} y_\gamma^{(i)} x_\alpha x_\beta x_\gamma \right) \pmod 8 \\ &= \sum_{\alpha}^{n} l_\alpha x_\alpha - \sum_{\alpha<\beta}^{n} 2 q_{\alpha,\beta} x_\alpha x_\beta + \sum_{\alpha<\beta<\gamma}^{n} 4 c_{\alpha,\beta,\gamma} x_\alpha x_\beta x_\gamma \pmod 8 \end{aligned} \tag{9.11}$$

where $l_\alpha, q_{\alpha,\beta}$ and $c_{\alpha,\beta,\gamma}$ are satisfying

$$\begin{aligned} l_\alpha &= \sum_{i=1}^{m} a_i y_\alpha^{(i)} \pmod 8, \\ q_{\alpha,\beta} &= \sum_{i=1}^{m} a_i y_\alpha^{(i)} y_\beta^{(i)} \pmod 4, \\ c_{\alpha,\beta,\gamma} &= \sum_{i=1}^{m} a_i y_\alpha^{(i)} y_\beta^{(i)} y_\gamma^{(i)} \pmod 2. \end{aligned} \tag{9.12}$$

Two weighted polynomials f and f', with associated coefficients $l_\alpha, q_{\alpha,\beta}, c_{\alpha,\beta,\gamma}$ and $l'_\alpha, q'_{\alpha,\beta}, c'_{\alpha,\beta,\gamma}$ respectively, are equal if and only if $l_\alpha = l'_\alpha, q_{\alpha,\beta} = q'_{\alpha,\beta}$ and $c_{\alpha,\beta,\gamma} = c'_{\alpha,\beta,\gamma}$ for all α, β, γ. Indeed, if these equalities are not satisfied, then we can find some vector $\boldsymbol{x}$ such that $f(\boldsymbol{x}) \neq f'(\boldsymbol{x})$. Therefore, two phase polynomials p and p' are implementing the same operator if and only if their associated weighted polynomials f and f' are equal. The problem of T-count optimization then consists in finding a phase polynomial p which implements a given weighted polynomial f with a minimal number of T gates. Notice that the coefficients of the weighted polynomial represented in Equation 9.10 all have the same parity: $l_\alpha \equiv q_{\alpha,\beta} \equiv c_{\alpha,\beta,\gamma} \equiv a \pmod 2$. And if a is even, then the associated rotation of angle $a\pi/4$ is a multiple of $\pi/2$ and can be implemented using only S gates. That is why a weighted polynomial f can be implemented using only $\{\text{CNOT}, S\}$ gates if and only if all its coefficients have an even parity, in such case we say that f is a Clifford weighted polynomial. Then, a weighted polynomial p with an associated parity table P is an implentation of a weighted polynomial f with an associated signature tensor $\mathcal{A}$ up to an operator implementable over the $\{\text{CNOT}, S\}$ gate set if and only if the equality

$$\mathcal{A}_{\alpha,\beta,\gamma} = |P_\alpha \wedge P_\beta \wedge P_\gamma| \pmod 2 \tag{9.13}$$

is satisfied for all α, β, γ satisfying $0 \leq \alpha \leq \beta \leq \gamma < n$, where n is the number of qubits. Throughout the paper, the notation $|\boldsymbol{v}|$ will be used to refer to the Hamming weight of the vector $\boldsymbol{v}$, and the symbol $\wedge$ will refer to the logical AND operation. Let f' be the weighted polynomial implemented by p, then f and f' have the same signature tensor, and f can be implemented by performing the synthesis of p and the synthesis of the Clifford operator associated with the Clifford weighted polynomial $f - f'$. The problem of finding a phase polynomial implementing a given weighted polynomial with a minimal number of T gates and up to an operator implementable over the $\{\text{CNOT}, S\}$ gate set can then be described by the following third order symmetric tensor rank decomposition (3-STR) problem [108].

Problem 4 (3-STR). *Let $\mathcal{A} \in \mathbb{Z}_2^{(n,n,n)}$ be a symmetric tensor such that*

$$\mathcal{A}_{\alpha,\beta,\gamma} = \mathcal{A}_{\alpha',\beta',\gamma'} \tag{9.14}$$

for all α, β, γ and α', β', γ' satisfying the set equality $\{\alpha, \beta, \gamma\} = \{\alpha', \beta', \gamma'\}$. Find a Boolean matrix P of size $n \times m$ such that

$$\mathcal{A}_{\alpha,\beta,\gamma} = |P_\alpha \wedge P_\beta \wedge P_\gamma| \pmod 2 \tag{9.15}$$

for all α, β, γ satisfying $0 \leq \alpha \leq \beta \leq \gamma < n$, with minimal m.

The T-count minimization problem in $\{\text{CNOT}, T, S\}$ circuits have first been shown to be equivalent to finding a minimum distance decoding in the order $n-4$ punctured Reed-Muller code of length $2^n - 1$ [106], noted $\mathcal{RM}(n-4, n)^*$, which is equivalent to the 3-STR problem [120]. The complexity class of the 3-STR problem is unknown, however, the related problem of finding the tensor rank of a tensor of order 3 is NP-complete [161]. And the more general problem of optimizing the number of T gates in a Clifford+T circuit is NP-hard [162].

An illustrative overview of the process we described for the optimization of the number of T gates in Hadamard-free circuits is provided in Figure 9.3. In Section 9.2, we present two algorithms attempting to solve the 3-STR problem and we prove that the number of T gates in the circuit produced by these algorithms is upper bounded by $(n^2 + n)/2 + 1$. We evaluate the perfomances of our algorithms in the benchmarks of Section 9.3.1. And we generalize our results for the optimization of the number of $R_Z(\pi/2^d)$ gates, where d is a non-negative integer, in Section 9.4.

9.2 T-count reduction algorithms

In this section we tackle the 3-STR problem as defined in Section 9.1. First, we propose an algorithm for this problem in Subsection 9.2.1 and we prove that the number of T gates in the solution produced by this algorithm is upper bounded by $n(n+1)/2+1$. Then, in Subsection 9.2.2, we show how the complexity of the `TODD` algorithm of Reference [108] can be reduced and we propose some modifications to this algorithm to improve its performances.

9.2.1 Third order homogeneous polynomials elimination algorithm

The key mechanism used by our algorithm for reducing the T-count is based on the following theorem.

Theorem 14. *Let P be a parity table of size $n \times m$ and $P' = P \oplus \boldsymbol{z}\boldsymbol{y}^T$ where $\boldsymbol{z}$ and $\boldsymbol{y}$ are vectors of size n and m respectively such that*

$$|\boldsymbol{y}| \equiv 0 \pmod 2 \tag{9.16}$$
$$|P_\alpha \wedge \boldsymbol{y}| \equiv 0 \pmod 2 \tag{9.17}$$
$$|P_\alpha \wedge P_\beta \wedge \boldsymbol{y}| \equiv 0 \pmod 2 \tag{9.18}$$

for all $0 \leq \alpha < \beta < n$. Then we have

$$|P'_\alpha \wedge P'_\beta \wedge P'_\gamma| \equiv |P_\alpha \wedge P_\beta \wedge P_\gamma| \pmod 2 \tag{9.19}$$

for all $0 \leq \alpha \leq \beta \leq \gamma < n$.

Proof. For all α, β, γ satisfying $0 \leq \alpha \leq \beta \leq \gamma < n$, we have:

$$\begin{aligned}
|P'_\alpha \wedge P'_\beta \wedge P'_\gamma| &= |(P_\alpha \oplus z_\alpha \boldsymbol{y}) \wedge (P_\beta \oplus z_\beta \boldsymbol{y}) \wedge (P_\gamma \oplus z_\gamma \boldsymbol{y})| \\
&= |[(P_\alpha \wedge P_\beta) \oplus z_\beta (P_\alpha \wedge \boldsymbol{y}) \oplus z_\alpha (P_\beta \wedge \boldsymbol{y}) \oplus z_\alpha z_\beta \boldsymbol{y}] \wedge (P_\gamma \oplus z_\gamma \boldsymbol{y})| \\
&\equiv |P_\alpha \wedge P_\beta \wedge P_\gamma| + z_\gamma |P_\alpha \wedge P_\beta \wedge \boldsymbol{y}| + z_\beta |P_\alpha \wedge P_\gamma \wedge \boldsymbol{y}| + z_\alpha |P_\beta \wedge P_\gamma \wedge \boldsymbol{y}| \quad (9.20) \\
&\quad + z_\beta z_\gamma |P_\alpha \wedge \boldsymbol{y}| + z_\alpha z_\gamma |P_\beta \wedge \boldsymbol{y}| + z_\alpha z_\beta |P_\gamma \wedge \boldsymbol{y}| + z_\alpha z_\beta z_\gamma |\boldsymbol{y}| \pmod 2 \\
&\equiv |P_\alpha \wedge P_\beta \wedge P_\gamma| \pmod 2
\end{aligned}$$

□

If the conditions of Theorem 14 are satisfied, then the set of columns selected by $\boldsymbol{y}$ are forming a weighted polynomial that can be divided into two weighted polynomials f and f', where f is a third order homogeneous weighted polynomial:

$$f(\boldsymbol{x}) = 4 \sum_{\alpha<\beta<\gamma}^{n} c_{\alpha,\beta,\gamma} x_\alpha x_\beta x_\gamma \pmod 8 \tag{9.21}$$

where $c_{\alpha,\beta,\gamma} = |P_\alpha \wedge P_\beta \wedge P_\gamma \wedge \boldsymbol{y}| \pmod 2$, and where f' is a Clifford weighted polynomial:

$$f'(\boldsymbol{x}) = \sum_{\alpha}^{n} l_\alpha x_\alpha + 2 \sum_{\alpha<\beta}^{n} q_{\alpha,\beta} x_\alpha x_\beta \pmod 8 \tag{9.22}$$

where $l_\alpha \in \mathbb{Z}_8$, $q_{\alpha,\beta} \in \mathbb{Z}_4$ and $l_\alpha \equiv q_{\alpha,\beta} \equiv 0 \pmod 2$. The unitary gate associated with the weighted polynomial f belongs to the $\mathcal{D}_3^C$ group: it is implementable using only CCZ gates [157]. It has already been shown that such weighted polynomials can be exploited to reduce the T-count, notably via the subadditivity theorem of Reference [157]. Let $U \in \mathcal{D}_3$ act on n qubits, in the following we define $\tau[U]$ as the optimal T-count to implement U without ancillary qubits:

$$\tau[U] = \min\{t \mid U = C_0 T_1 C_1 \dots T_t C_t, \{C_0, \dots, C_t\} \in \mathcal{C}_n^*\} \tag{9.23}$$

where $\mathcal{C}_n^*$ is the subgroup of Clifford operators which can be implemented with CNOT and S gates. The subadditivity theorem of Reference [157] states that if $\tau[U_1] \equiv 1 \pmod 2$ and $\tau[U_2] > 0$, then $\tau[U_1 \otimes U_2] < \tau[U_1] + \tau[U_2]$ where $U_1 \in \mathcal{D}_3^C$ and $U_2 \in \mathcal{D}_3$. Based on Theorem 14, we can actually remove the condition $\tau[U_1] \equiv 1 \pmod 2$, which gives the following theorem.

Theorem 15 (Subadditivity theorem)**.** *Let $U_1 \in \mathcal{D}_3^C$, and $U_2 \in \mathcal{D}_3$. If $\tau[U_1], \tau[U_2] > 0$, then $\tau[U_1 \otimes U_2] < \tau[U_1] + \tau[U_2]$.*

Proof. Let $W = \begin{bmatrix} P & Q \end{bmatrix}$ be a parity table such that P and Q are the parity tables associated with the implementation of U_1 and U_2 and which have $\tau[U_1]$ and $\tau[U_2]$ columns respectively. Then, because $U_1 \in \mathcal{D}_3^C$, P satisfies the following equations:

$$|P_\alpha| \equiv 0 \pmod 2 \tag{9.24}$$

$$|P_\alpha \wedge P_\beta| \equiv 0 \pmod 2 \tag{9.25}$$

for all α, β. Let $\boldsymbol{z} = P_{:,i} \oplus Q_{:,j}$ for any i and j satisfying $0 \leq i < \tau[U_1]$, $0 \leq j < \tau[U_2]$, where $P_{:,i}$ denotes the ith column of the parity table P. And let P' be a parity table such that

$$P' = \begin{cases} P \oplus \boldsymbol{z}\boldsymbol{1}^T & \text{if } \tau[U_1] \equiv 0 \pmod 2, \\ \begin{bmatrix} P \oplus \boldsymbol{z}\boldsymbol{1}^T & \boldsymbol{z} \end{bmatrix} & \text{otherwise.} \end{cases}$$

Then, as stated by Theorem 14, the parity table P' satisfies

$$|P'_\alpha \wedge P'_\beta \wedge P'_\gamma| \equiv |P_\alpha \wedge P_\beta \wedge P_\gamma| \pmod 2 \tag{9.26}$$

for all α, β, γ. And so the parity table $W' = \begin{bmatrix} P' & Q \end{bmatrix}$, which has at most one more column than W, also satisfies

$$|W'_\alpha \wedge W'_\beta \wedge W'_\gamma| \equiv |W_\alpha \wedge W_\beta \wedge W_\gamma| \pmod 2 \tag{9.27}$$

for all α, β, γ. However, we can notice that $P'_{:,i} = Q_{:,j}$. Therefore, by removing these two columns from W' Equation 9.27 still holds and W' has at least one less column than W. The parity table W' implements the unitary $U_1 \otimes U_2$ up to a Clifford operator and has at most $\tau[U_1] + \tau[U_2] - 1$ columns, thus we have $\tau[U_1 \otimes U_2] \leq \tau[U_1] + \tau[U_2] - 1 < \tau[U_1] + \tau[U_2]$. □

Based on this subadditivity theorem and on Theorem 14, we can derive the following upper bound on the number of T gates in a $\{\text{CNOT}, S, T\}$ circuit.

Theorem 16. *The number of T gates in an n-qubits $\{\text{CNOT}, T, S\}$ circuit can be upper bounded by*

$$2\lfloor (n^2+n)/4 \rfloor + 1 \leq (n^2+n)/2 + 1 \tag{9.28}$$

in polynomial time.

Proof. Let U be a unitary gate implementable by a $\{\text{CNOT}, S, T\}$ gate set, let P be a parity table of size $n \times m$ which implements U up to an operator implementable over the $\{\text{CNOT}, S\}$ gate set, and let L be a matrix with rows labelled by $(\alpha\beta)$ such that

$$L_{\alpha\beta} = P_\alpha \wedge P_\beta \tag{9.29}$$

for all α, β satisfying $0 \leq \alpha \leq \beta < n$. If P has strictly more than $(n^2+n)/2 + 1$ columns then we can necessarily find a non-zero vector $\boldsymbol{y}$ satisfying $L\boldsymbol{y} = \boldsymbol{0}$ and $\boldsymbol{y} \neq \boldsymbol{1}$ because L has $(n^2+n)/2$ rows. Note that such vector $\boldsymbol{y}$ necessarily satisfies Equations 9.17 and 9.18 of Theorem 14. We can then divide P into two non-empty parity tables $P^{(1)}$ and $P^{(2)}$ where the column $P_{:,i}$ belongs to $P^{(1)}$ if and only if $y_i = 1$ and to $P^{(2)}$ otherwise. The parity tables $P^{(1)}$ and $P^{(2)}$ are implementations of some unitary gates $U_1 \in \mathcal{D}_3^C$ and $U_2 \in \mathcal{D}_3$ respectively. The subadditivity theorem (Theorem 15) can then be exploited to reduce the number of columns of P. Let $\boldsymbol{z} = P_{:,i} \oplus P_{:,j}$ where i and j are satisfying $y_i = 1$ and $y_j = 0$, and let

$$P' = \begin{cases} P \oplus \boldsymbol{z}\boldsymbol{y}^T & \text{if } |\boldsymbol{y}| \equiv 0 \pmod 2, \\ \begin{bmatrix} P \oplus \boldsymbol{z}\boldsymbol{y}^T & \boldsymbol{z} \end{bmatrix} & \text{otherwise.} \end{cases}$$

By Theorem 14, P' implements the same unitary gate as P up to an operator implementable over the $\{\text{CNOT}, S\}$ gate set. The parity table P' has at most one more column than P and we have $P'_{:,i} = P'_{:,j}$. Therefore, the columns i and j can be removed from P', which entails that P' has at least one less column than P. We showed that if the number of columns of P is strictly greater than $(n^2+n)/2+1$, then the number of columns of P can be reduced by at least one in polynomial time. Moreover, if the number of columns of P is equal to $(n^2+n)/2+1$ and is even, then we can necessarily find a non-zero vector $\boldsymbol{y}$ satisfying $L\boldsymbol{y} = \boldsymbol{0}$ because L has $(n^2+n)/2$ rows. If there exist i such that $y_i = 0$ then we can reduce the number of columns of P as described above. Otherwise we must have $|\boldsymbol{y}| \equiv 0 \pmod 2$, and so $\boldsymbol{y}$ satisfies the Equations of Theorem 14. Therefore, if $P' = P \oplus \boldsymbol{z}\boldsymbol{y}^T$ where $\boldsymbol{z}$ is equal to the ith column of P for any i, then P' implements the same unitary gate as P up to an operator implementable over the $\{\text{CNOT},$

$S\}$ gate set. The parity table P' has the same number of columns as P but its ith column is equal to the null vector and can therefore be removed, which leads to a parity table containing $(n^2+n)/2$ columns. The polynomial-time procedure described above to reduce the number of columns of P can be repeated until P has a number of columns lower or equal to

$$2\lfloor (n^2+n)/4 \rfloor + 1 \leq (n^2+n)/2 + 1 \tag{9.30}$$

□

Note that, for $n > 5$, this upper bound is better than the previously best known upper bound of $(n^2+3n-14)/2$ [157]. The proof of Theorem 16 provides a straightforward algorithm for achieving this upper bound in polynomial time. We propose some improvements regarding this approach by providing an algorithm whose pseudo-code is given in Algorithm 13.

The algorithm starts by computing the set Z which contains all the vectors $\boldsymbol{z}$ which can potentially be used to transform P via Theorem 14 and reduce its number of columns:

$$Z = \{P_{:,i} \oplus P_{:,j} \mid 0 \leq i < j < m\} \cup \{P_{:,i} \mid 0 \leq i < m\}. \tag{9.31}$$

For each vector $\boldsymbol{z} \in Z$, the algorithm also computes the set $S^{(\boldsymbol{z})}$ of pairs of indices $\{i, j\}$ satisfying $P_{:,i} \oplus P_{:,j} = \boldsymbol{z}$ in the case where $i \neq j$ or satisfying $P_{:,i} = \boldsymbol{z}$ in the case where $i = j$. Let L be a matrix with rows labelled by $(\alpha\beta)$ such that

$$L_{\alpha\beta} = P_\alpha \wedge P_\beta \tag{9.32}$$

for all $\alpha < \beta$. In order to exploit the subadditivity theorem, the vector $\boldsymbol{y}$ must satisfy $L\boldsymbol{y} = \boldsymbol{0}$, $\boldsymbol{y} \neq \boldsymbol{0}$ and $\boldsymbol{y} \neq \boldsymbol{1}$. However, in the case where $\boldsymbol{y} = \boldsymbol{1}$ and $|\boldsymbol{y}| \equiv 0 \pmod 2$ the number of columns of P can still be reduced. Indeed, in such case the vector $\boldsymbol{y}$ satisfies all the condition of Theorem 14, therefore the parity table P' defined as $P' = P \oplus P_{:,i}\boldsymbol{1}^T$ for any i is equivalent to P and its ith column is equal to the null vector and can therefore be removed, which reduces the number of columns by one. To summarize, the number of columns of P can be reduced if $\boldsymbol{y}$ satisfies $L\boldsymbol{y} = \boldsymbol{0}$ and $\boldsymbol{y} \neq \boldsymbol{0}$ and $\boldsymbol{y} \neq \boldsymbol{1}$ or $|\boldsymbol{y}| \equiv 0 \pmod 2$. If no such $\boldsymbol{y}$ is found, Algorithm 13 will stop and return P. Otherwise, the algorithm will select a vector $\boldsymbol{z}$ such that the number of duplicated columns plus the number of all-zero columns in the parity table $P' = P \oplus \boldsymbol{z}\boldsymbol{y}^T$ is maximized. This number corresponds to the number of columns that can be removed from P if $\boldsymbol{z}$ is chosen (minus one in the case where $|\boldsymbol{y}| \equiv 1 \pmod 2$), and is given by the following objective function:

$$-|\boldsymbol{y}| \pmod 2 + \sum_{\{i,j\} \in S^{(\boldsymbol{z})}} \begin{cases} 2(y_i \oplus y_j) & \text{if } i \neq j, \\ y_i + 2(y_i \oplus 1)(|\boldsymbol{y}| \pmod 2) & \text{if } i = j. \end{cases} \tag{9.33}$$

The case where $i \neq j$ is straightforward, as the ith column will be equal to the jth column of P' if and only if $y_i \oplus y_j = 1$, which results in the removal of 2 columns. In the case where $i = j$, the equation

$$y_i + 2(y_i \oplus 1)(|\boldsymbol{y}| \pmod 2) \tag{9.34}$$

is equal to 1 if $y_i = 1$ because the ith column of P' is the null vector, leading to the removal of 1 column. Otherwise, if $y_i = 0$ and $|\boldsymbol{y}|$ is odd, then Equation 9.34 is equal to 2 because the ith column of P' will be equal the $\boldsymbol{z}$ column vector that is added to P' in the case where $|\boldsymbol{y}|$ is odd, resulting in the removal of 2 columns. Once the vector $\boldsymbol{z}$ maximizing this objective function has

Algorithm 13: Third order homogeneous polynomials elimination algorithm

Input: A parity table P of size $n \times m$.

Output: An equivalent parity table with an optimized number of columns.

procedure TOHPE(P)
  $Z \leftarrow \{P_{:,i} \oplus P_{:,j} \mid 0 \leq i < j < m\} \cup \{P_{:,i} \mid 0 \leq i < m\}$
  forall $\boldsymbol{z} \in Z$ **do**
    $S^{(\boldsymbol{z})} \leftarrow \{\{i,j\} \mid P_{:,i} \oplus P_{:,j} = \boldsymbol{z}\} \cup \{\{i,i\} \mid P_{:,i} = \boldsymbol{z}\}$
  end
  $L \leftarrow$ matrix whose rows are forming the set $\{P_i \wedge P_j \mid 0 \leq i \leq j < n\}$
  if $\nexists \boldsymbol{y}$ such that $L\boldsymbol{y} = \boldsymbol{0}$ and $\boldsymbol{y} \neq \boldsymbol{0}$ and ($\boldsymbol{y} \neq \boldsymbol{1}$ or $|\boldsymbol{y}| \equiv 0 \pmod 2$) **then**
    return P
  end
  $\boldsymbol{y} \leftarrow$ any vector such that $L\boldsymbol{y} = \boldsymbol{0}$ and $\boldsymbol{y} \neq \boldsymbol{0}$ and ($\boldsymbol{y} \neq \boldsymbol{1}$ or $|\boldsymbol{y}| \equiv 0 \pmod 2$)
  $\boldsymbol{z} \leftarrow \underset{\boldsymbol{z} \in Z}{\operatorname{argmax}} \left\{ -|\boldsymbol{y}| \pmod 2 + \sum_{\{i,j\} \in S^{(\boldsymbol{z})}} 2(y_i \oplus y_j) + \delta_{ij}\left[y_i + 2(y_i \oplus 1)(|\boldsymbol{y}| \pmod 2)\right] \right\}$
  $P' \leftarrow P \oplus \boldsymbol{z}\boldsymbol{y}^T$
  if $|\boldsymbol{y}| \equiv 1 \pmod 2$ **then**
    $P' \leftarrow \begin{bmatrix} P' & \boldsymbol{z} \end{bmatrix}$
  end
  $P' \leftarrow P'$ with all its duplicated and all-zero columns removed
  return TOHPE(P')

been found, Algorithm 13 then computes the new parity table P' which contains fewer columns, and performs a recursive call.

A variant of this algorithm could consist in finding both vectors $\boldsymbol{z}$ and $\boldsymbol{y}$ that are maximizing the objective function. However, our experiments on this approach showed that it significantly increases the complexity of the algorithm for a rather marginal gain in the number of T gates. The performances of Algorithm 13 are evaluated in Section 9.3.1.

Complexity analysis. The set Z can be computed with $\mathcal{O}(nm^2)$ operations, and all the sets $S^{(\boldsymbol{z})}$ can be created with the same time complexity. The matrix L has $\mathcal{O}(n^2)$ rows and a number of columns that is at most equal to m. Therefore performing a Gaussian elimination to compute a generating set of the right nullspace of L implies a complexity of $\mathcal{O}(n^2m^2)$. The vector $\boldsymbol{z}$ satisfying the argmax function can be computed in $\mathcal{O}(m^2)$ operations because the union of all the $S^{(\boldsymbol{z})}$ sets contains $\mathcal{O}(m^2)$ elements. Updating the parity table P induces $\mathcal{O}(nm)$ operations. And the algorithm is performing no more than m recursive calls. Thus, the overall complexity of Algorithm 13 is $\mathcal{O}(n^2m^3)$.

9.2.2 Improving the TODD algorithm

In this section we show how the TODD algorithm proposed in Reference [108] can be improved. We first describe the algorithm and demonstrate how its complexity can be reduced. We then propose a modified version of this algorithm to improve its performances.

The key mechanism of the TODD algorithm rests on the following theorem, which was first proven in Reference [108]. We provide its proof for completeness.

Theorem 17. *Let P be a parity table of size $n \times m$ and $P' = P \oplus \boldsymbol{z}\boldsymbol{y}^T$ where $\boldsymbol{z}$ and $\boldsymbol{y}$ are vectors of size n and m respectively such that*

$$|\boldsymbol{y}| \equiv 0 \pmod 2 \tag{9.35}$$
$$|P_\alpha \wedge \boldsymbol{y}| \equiv 0 \pmod 2 \tag{9.36}$$
$$|[z_\alpha(P_\beta \wedge P_\gamma) \oplus z_\beta(P_\alpha \wedge P_\gamma) \oplus z_\gamma(P_\alpha \wedge P_\beta)] \wedge \boldsymbol{y}| \equiv 0 \pmod 2 \tag{9.37}$$

for all $0 \leq \alpha < \beta < \gamma < n$. Then we have

$$|P'_\alpha \wedge P'_\beta \wedge P'_\gamma| \equiv |P_\alpha \wedge P_\beta \wedge P_\gamma| \pmod 2 \tag{9.38}$$

for all $0 \leq \alpha \leq \beta \leq \gamma < n$.

Proof. Analogously to Equation 9.20 and using Equations 9.35 and 9.36, we have

$$\begin{aligned}
&|P'_\alpha \wedge P'_\beta \wedge P'_\gamma| \\
&\equiv |P_\alpha \wedge P_\beta \wedge P_\gamma| + z_\gamma|P_\alpha \wedge P_\beta \wedge \boldsymbol{y}| + z_\beta|P_\alpha \wedge P_\gamma \wedge \boldsymbol{y}| + z_\alpha|P_\beta \wedge P_\gamma \wedge \boldsymbol{y}| \\
&\quad + z_\beta z_\gamma|P_\alpha \wedge \boldsymbol{y}| + z_\alpha z_\gamma|P_\beta \wedge \boldsymbol{y}| + z_\alpha z_\beta|P_\gamma \wedge \boldsymbol{y}| + z_\alpha z_\beta z_\gamma|\boldsymbol{y}| \pmod 2 \\
&\equiv |P_\alpha \wedge P_\beta \wedge P_\gamma| + |[z_\alpha(P_\beta \wedge P_\gamma) \oplus z_\beta(P_\alpha \wedge P_\gamma) \oplus z_\gamma(P_\alpha \wedge P_\beta)] \wedge \boldsymbol{y}| \pmod 2
\end{aligned} \tag{9.39}$$

for all α, β, γ satisfying $0 \leq \alpha \leq \beta \leq \gamma < n$. In the case where $\alpha \neq \beta \neq \gamma$, we obtain

$$|P'_\alpha \wedge P'_\beta \wedge P'_\gamma| \equiv |P_\alpha \wedge P_\beta \wedge P_\gamma| \pmod 2 \tag{9.40}$$

by using Equation 9.37. In the case where $\alpha = \beta$, we obtain

$$\begin{aligned}
|P'_\alpha \wedge P'_\beta \wedge P'_\gamma| &\equiv |P_\alpha \wedge P_\beta \wedge P_\gamma| + |[z_\alpha(P_\alpha \wedge P_\gamma) \oplus z_\alpha(P_\alpha \wedge P_\gamma) \oplus z_\gamma P_\alpha] \wedge \boldsymbol{y}| \pmod 2 \\
&\equiv |P_\alpha \wedge P_\beta \wedge P_\gamma| \pmod 2
\end{aligned} \tag{9.41}$$

by using Equation 9.36. Finally, in the case where $\alpha = \beta = \gamma$, we obtain

$$\begin{aligned}
|P'_\alpha \wedge P'_\beta \wedge P'_\gamma| &\equiv |P_\alpha \wedge P_\beta \wedge P_\gamma| + |[z_\alpha P_\alpha \oplus z_\alpha P_\alpha \oplus z_\alpha P_\alpha] \wedge \boldsymbol{y}| \pmod 2 \\
&\equiv |P_\alpha \wedge P_\beta \wedge P_\gamma| \pmod 2
\end{aligned} \tag{9.42}$$

by using Equation 9.36. □

The `TODD` algorithm exploits this theorem for optimizing the number of T gates as follows. Let P be a parity table of size $n \times m$, let $\boldsymbol{z} = P_{:,i} \oplus P_{:,j}$ where $i \neq j$ and let χ be a matrix that contains a row equal to P_α for every α and a row equal to

$$z_\alpha(P_\beta \wedge P_\gamma) \oplus z_\beta(P_\alpha \wedge P_\gamma) \oplus z_\gamma(P_\alpha \wedge P_\beta)$$

for every triple α, β, γ satisfying $0 \leq \alpha < \beta < \gamma < n$. A vector $\boldsymbol{y}$ satisfying the conditions 9.36 and 9.37 of Theorem 17 can be found by computed the nullspace of χ, i.e. if $\boldsymbol{y}$ satisfies $\chi\boldsymbol{y} = \boldsymbol{0}$ then $\boldsymbol{y}$ also satisfies the conditions 9.36 and 9.37 of Theorem 17. If $\boldsymbol{y}$ satisfies $\chi\boldsymbol{y} = \boldsymbol{0}$ and $y_i \oplus y_j = 1$, then the number of columns of P can be reduced. Let

$$P' = \begin{cases} P \oplus \boldsymbol{z}\boldsymbol{y}^T & \text{if } |\boldsymbol{y}| \equiv 0 \pmod 2, \\ \begin{bmatrix} P \oplus \boldsymbol{z}\boldsymbol{y}^T & \boldsymbol{z} \end{bmatrix} & \text{otherwise.} \end{cases}$$

Then, as stated by Theorem 17, P and P' are equivalent. Moreover, we have $P'_{:,i} = P'_{:,j}$ because $\boldsymbol{z} = P_{:,i} \oplus P_{:,j}$, therefore these two columns can be removed from P', which results in P' having at least one less column than P.

If there doesn't exist any $\boldsymbol{y}$ satisfying $\chi\boldsymbol{y} = \boldsymbol{0}$ and $y_i \oplus y_j = 1$, then a different $\boldsymbol{z}$ vector can be considered. In the worst case the `TODD` algorithm performs this search for all the vectors $\boldsymbol{z}$ in the set $\{P_{:,i} \oplus P_{:j} \mid 0 \leq i < j < n\}$, which contains $\mathcal{O}(m^2)$ elements. The matrix χ is composed of m columns and $\mathcal{O}(n^3)$ rows. Performing a Gaussian elimination in order to find some $\boldsymbol{y}$ satisfying $\chi\boldsymbol{y} = \boldsymbol{0}$ induces a complexity of $\mathcal{O}(n^3m^2)$. This procedure must be performed for the $\mathcal{O}(m^2)$ $\boldsymbol{z}$ vectors evaluated. And everytime a simplification is found the algorithm restart the same procedure on the new parity table $P' = P \oplus \boldsymbol{z}\boldsymbol{y}^T$, this happens at most m times. Thus, the overall complexity of the `TODD` algorithm is $\mathcal{O}(n^3m^5)$. The important worst-case complexity of the `TODD` algorithm makes it rapidly impractical for instances of increasing size. We will demonstrate how this complexity can be reduced to $\mathcal{O}(n^4m^3)$ by providing an alternative and more efficient way of checking whether or not there exists a vector $\boldsymbol{y}$ satisfying the conditions of Theorem 17, and therefore avoiding the expensive Gaussian elimination required to solve the system $\chi\boldsymbol{y} = \boldsymbol{0}$.

First, we can notice that Equation 9.37 of Theorem 17 can be greatly simplified in some cases, depending on the value of $\boldsymbol{z}$. For instance, if $z_\alpha = 1$ for some α and $z_\beta = 0$ for all $\beta \neq \alpha$, then Equation 9.37 is equal to

$$|z_\alpha(P_\beta \wedge P_\gamma) \wedge \boldsymbol{y}| \equiv 0 \pmod 2. \tag{9.43}$$

In this specific case, a vector $\boldsymbol{y}$ satisfies Equations 9.36 and 9.37 of Theorem 17 if it satisfies $\chi\boldsymbol{y} = \boldsymbol{0}$ where χ contains a row equal to Equation 9.36 for all α and a row equal to Equation 9.43 for all α, β, γ satisfying $\alpha \neq \beta \neq \gamma$, $\beta < \gamma$ and $z_\alpha = 1$. In total, χ contains only $\mathcal{O}(n^2)$ rows instead of $\mathcal{O}(n^3)$ rows in the original `TODD` algorithm, and so the nullspace of χ can be computed in $\mathcal{O}(n^2m^2)$ operations. In the more general case where $|\boldsymbol{z}| > 1$, we can perform a change of basis to always end up in this specific case where $|\boldsymbol{z}| = 1$. A change of basis induces a complexity of $\mathcal{O}(nm)$ because $\mathcal{O}(n)$ additions must be performed between the rows of P. This must be done for all the $\mathcal{O}(m^2)$ $\boldsymbol{z}$ vectors evaluated, and the whole procedure is repeated $\mathcal{O}(m)$ times. Thus, this version of the `TODD` algorithm has a complexity of $\mathcal{O}(n^2m^5)$, which is better than the $\mathcal{O}(n^3m^5)$ complexity of the original `TODD` algorithm.

Instead of performing a change of basis, we can rely on the following theorem to efficiently find the vectors $\boldsymbol{z}$ and $\boldsymbol{y}$ satisfying the Equations 9.36 and 9.37 of Theorem 17.

Theorem 18. *Let P be a parity table of size $n \times m$, and let $\boldsymbol{z}$ and $\boldsymbol{y}$ be vectors of size n and m respectively and such that $|\boldsymbol{y}| \equiv 0 \pmod 2$. Let L and X be matrices with rows labelled by $(\alpha\beta)$ such that*

$$L_{\alpha\beta} = P_\alpha \wedge P_\beta \tag{9.44}$$

and

$$X_{\alpha\beta,\gamma} = z_\alpha\delta_{\beta\gamma} \oplus z_\beta\delta_{\alpha\gamma} \tag{9.45}$$

for all α, β, γ satisfying $0 \leq \alpha \leq \beta < n$ and $0 \leq \gamma < n$, and where δ is the Kronecker delta defined as follows:

$$\delta_{\alpha\beta} = \begin{cases} 0 & \text{if } \alpha \neq \beta, \\ 1 & \text{if } \alpha = \beta. \end{cases} \tag{9.46}$$

There exists $\boldsymbol{y}'$ *such that* $L\boldsymbol{y} \oplus X\boldsymbol{y}' = \boldsymbol{0}$ *if and only if the following conditions are satisfied:*

$$|P_\alpha \wedge \boldsymbol{y}| \equiv 0 \pmod 2 \tag{9.47}$$

$$|[z_\alpha(P_\beta \wedge P_\gamma) \oplus z_\beta(P_\alpha \wedge P_\gamma) \oplus z_\gamma(P_\alpha \wedge P_\beta)] \wedge \boldsymbol{y}| \equiv 0 \pmod 2 \tag{9.48}$$

for all $0 \leq \alpha \leq \beta \leq \gamma < n$.

Proof. In the following we will use the labels $(\beta\alpha)$ and $(\alpha\beta)$ to refer to the same unique row, for instance $L_{\beta\alpha} = L_{\alpha\beta}$. For all α we have $X_{\alpha\alpha} = \boldsymbol{0}$, which implies that

$$L_{\alpha\alpha}\boldsymbol{y} \oplus X_{\alpha\alpha}\boldsymbol{y}' = L_{\alpha\alpha}\boldsymbol{y} \equiv |P_\alpha \wedge \boldsymbol{y}| \pmod 2 \tag{9.49}$$

for all α. Therefore we have

$$L_{\alpha\alpha}\boldsymbol{y} \oplus X_{\alpha\alpha}\boldsymbol{y}' = 0 \iff |P_\alpha \wedge \boldsymbol{y}| \equiv 0 \pmod 2 \tag{9.50}$$

for all α. It remains to show that there exists $\boldsymbol{y}'$ satisfying $L_{\alpha\beta}\boldsymbol{y} \oplus X_{\alpha\beta}\boldsymbol{y}' = 0$ for all $\alpha \neq \beta$ if and only if

$$|[z_\alpha(P_\beta \wedge P_\gamma) \oplus z_\beta(P_\alpha \wedge P_\gamma) \oplus z_\gamma(P_\alpha \wedge P_\beta)] \wedge \boldsymbol{y}| \equiv 0 \pmod 2 \tag{9.51}$$

holds for all $0 \leq \alpha < \beta < \gamma < n$.

In the case where $|\boldsymbol{z}| = 1$, let α be such that $z_\alpha = 1$. Then, for such vector $\boldsymbol{z}$, Equation 9.51 is equivalent to

$$|P_\beta \wedge P_\gamma \wedge \boldsymbol{y}| \equiv 0 \pmod 2 \tag{9.52}$$

for all β, γ satisfying $0 \leq \beta < \gamma < n$ and $\alpha \neq \beta \neq \gamma$. We also have $X_{\beta\gamma}\boldsymbol{y}' = 0$ for any $\boldsymbol{y}'$ and for all β, γ satisfying $\alpha \neq \beta \neq \gamma$ because $X_{\beta\gamma} = \boldsymbol{0}$ for all β, γ satisfying $z_\beta = 0, z_\gamma = 0$. We then have

$$L_{\beta\gamma}\boldsymbol{y} \oplus X_{\beta\gamma}\boldsymbol{y}' = L_{\beta\gamma}\boldsymbol{y} \equiv |P_\beta \wedge P_\gamma \wedge \boldsymbol{y}| \pmod 2 \tag{9.53}$$

for all β, γ satisfying $0 \leq \beta < \gamma < n$ and $\alpha \neq \beta \neq \gamma$. And therefore

$$L_{\beta\gamma}\boldsymbol{y} \oplus X_{\beta\gamma}\boldsymbol{y}' = \boldsymbol{0} \iff |P_\beta \wedge P_\gamma \wedge \boldsymbol{y}| \equiv 0 \pmod 2 \tag{9.54}$$

for all β, γ satisfying $0 \leq \beta < \gamma < n$ and $\alpha \neq \beta \neq \gamma$. Moreover, the definition of the matrix X implies the following equality for all β, γ:

$$X_{\beta\gamma}\boldsymbol{y}' = y'_\beta z_\gamma \oplus y'_\gamma z_\beta \tag{9.55}$$

which entails

$$L_{\alpha\beta}\boldsymbol{y} \oplus X_{\alpha\beta}\boldsymbol{y}' = L_{\alpha\beta}\boldsymbol{y} \oplus y'_\beta \tag{9.56}$$

for all β satisfying $\alpha \neq \beta$. Therefore, if $\boldsymbol{y}'$ satisfies $y'_\beta = L_{\alpha\beta}\boldsymbol{y}$ for all β such that $\alpha \neq \beta$, then $L\boldsymbol{y} \oplus X\boldsymbol{y}' = \boldsymbol{0}$ and so Theorem 18 is true in the case where $|\boldsymbol{z}| = 1$.

Let B be a full rank binary matrix of size $n \times n$ which represents a change of basis, and let $\tilde{L}$ and $\tilde{X}$ be matrices constructed in the same way as L and X but with respect to BP and $B\boldsymbol{z}$. The proof of Theorem 18 can be completed by proving the two following propositions. First, if $\boldsymbol{y}$ satisfies Equations 9.47 and 9.48 for P and $\boldsymbol{z}$ then $\boldsymbol{y}$ also satisfies Equations 9.47 and 9.48 for BP and $B\boldsymbol{z}$. Second, if there exists $\boldsymbol{y}'$ such that $L\boldsymbol{y} \oplus X\boldsymbol{y}' = \boldsymbol{0}$ then there exists $\tilde{\boldsymbol{y}}'$ such that $\tilde{L}\boldsymbol{y} \oplus \tilde{X}\tilde{\boldsymbol{y}}' = \boldsymbol{0}$. Indeed, if these two propositions are true, then if $\boldsymbol{y}$ satisfies Equations 9.47 and 9.48 for P and $\boldsymbol{z}$ it also satisfies Equations 9.47 and 9.48 for BP and $B\boldsymbol{z}$, where B is chosen such that $|B\boldsymbol{z}| = 1$. Then, as previously demonstrated, there exists a vector $\tilde{\boldsymbol{y}}'$ such that $\tilde{L}\boldsymbol{y} \oplus \tilde{X}\tilde{\boldsymbol{y}}' = \boldsymbol{0}$, which would imply that there exists a vector $\boldsymbol{y}'$ such that $L\boldsymbol{y} \oplus X\boldsymbol{y}' = \boldsymbol{0}$.

Let $\tilde{\boldsymbol{z}}$ and $\tilde{P}$ be such that $\tilde{z}_\alpha = z_\alpha \oplus z_\beta$ and $\tilde{P}_\alpha = P_\alpha \oplus P_\beta$ for some fixed α, β satisfying $\alpha \neq \beta$ and $\tilde{z}_\gamma = z_\gamma$, $\tilde{P}_\gamma = P_\gamma$ for all γ such that $\gamma \neq \alpha$. If $\boldsymbol{y}$ satisfies Equations 9.47 and 9.48, then we have

$$\begin{aligned} |\tilde{P}_\alpha \wedge \boldsymbol{y}| &= |(P_\alpha \oplus P_\beta) \wedge \boldsymbol{y}| \\ &\equiv |P_\alpha \wedge \boldsymbol{y}| + |P_\beta \wedge \boldsymbol{y}| \pmod 2 \\ &\equiv 0 \pmod 2 \end{aligned} \tag{9.57}$$

and

$$\begin{aligned} &|[\tilde{z}_\alpha(\tilde{P}_\beta \wedge \tilde{P}_\gamma) \oplus \tilde{z}_\beta(\tilde{P}_\alpha \wedge \tilde{P}_\gamma) \oplus \tilde{z}_\gamma(\tilde{P}_\alpha \wedge \tilde{P}_\beta)] \wedge \boldsymbol{y}| \\ &= |[(z_\alpha \oplus z_\beta)(P_\beta \wedge P_\gamma) \oplus z_\beta((P_\alpha \oplus P_\beta) \wedge P_\gamma) \oplus z_\gamma((P_\alpha \oplus P_\beta) \wedge P_\beta)] \wedge \boldsymbol{y}| \\ &= |[z_\alpha(P_\beta \wedge P_\gamma) \oplus z_\beta(P_\beta \wedge P_\gamma) \oplus z_\beta(P_\alpha \wedge P_\gamma) \oplus z_\beta(P_\beta \wedge P_\gamma) \oplus z_\gamma(P_\alpha \wedge P_\beta) \oplus z_\gamma P_\beta] \wedge \boldsymbol{y}| \\ &\equiv |[z_\alpha(P_\beta \wedge P_\gamma) \oplus z_\beta(P_\alpha \wedge P_\gamma) \oplus z_\gamma(P_\alpha \wedge P_\beta)] \wedge \boldsymbol{y}| + z_\gamma|P_\beta \wedge \boldsymbol{y}| \pmod 2 \\ &\equiv 0 \pmod 2 \end{aligned} \tag{9.58}$$

and so $\boldsymbol{y}$ also satisfies Equations 9.47 and 9.48 for $\tilde{P}, \tilde{\boldsymbol{z}}$.

Let $\tilde{L}$ and $\tilde{X}$ be matrices constructed in the same way as L and X but with respect to $\tilde{P}$ and $\tilde{\boldsymbol{z}}$. Let's assume that $L\boldsymbol{y} \oplus X\boldsymbol{y}' = \boldsymbol{0}$, we will show that it implies $\tilde{L}\boldsymbol{y} \oplus \tilde{X}\tilde{\boldsymbol{y}}' = \boldsymbol{0}$, where $\tilde{\boldsymbol{y}}'$ satisfies $\tilde{y}'_\alpha = y'_\alpha \oplus y'_\beta$ and $\tilde{y}'_\gamma = y'_\gamma$ for all γ such that $\gamma \neq \alpha$. By using Equation 9.55 we can deduce that

$$\tilde{X}_{\gamma\gamma'}\tilde{\boldsymbol{y}}' = \tilde{y}'_\gamma \tilde{z}_{\gamma'} \oplus \tilde{y}'_{\gamma'} \tilde{z}_\gamma = y'_\gamma z_{\gamma'} \oplus y'_{\gamma'} z_\gamma = X_{\gamma\gamma'}\boldsymbol{y}' \tag{9.59}$$

for all γ, γ' satisfying $\gamma \neq \alpha$ and $\gamma' \neq \alpha$. This entails

$$\tilde{L}_{\gamma\gamma'}\boldsymbol{y} \oplus \tilde{X}_{\gamma\gamma'}\tilde{\boldsymbol{y}}' = L_{\gamma\gamma'}\boldsymbol{y} \oplus X_{\gamma\gamma'}\boldsymbol{y}' = 0 \tag{9.60}$$

for all γ, γ' satisfying $\gamma \neq \alpha$ and $\gamma' \neq \alpha$. Furthermore, for all γ such that $\gamma \neq \alpha$, we have

$$\begin{aligned} \tilde{L}_{\alpha\gamma}\boldsymbol{y} &\equiv |\tilde{P}_\alpha \wedge \tilde{P}_\gamma \wedge \boldsymbol{y}| \pmod 2 \\ &\equiv |(P_\alpha \oplus P_\beta) \wedge P_\gamma \wedge \boldsymbol{y}| \pmod 2 \\ &\equiv |P_\alpha \wedge P_\gamma \wedge \boldsymbol{y}| + |P_\beta \wedge P_\gamma \wedge \boldsymbol{y}| \pmod 2 \\ &\equiv L_{\alpha\gamma}\boldsymbol{y} + L_{\beta\gamma}\boldsymbol{y} \pmod 2 \\ &\equiv X_{\alpha\gamma}\boldsymbol{y}' + X_{\beta\gamma}\boldsymbol{y}' \pmod 2 \end{aligned} \tag{9.61}$$

which entails

$$\begin{aligned} \tilde{L}_{\alpha\gamma}\boldsymbol{y} \oplus \tilde{X}_{\alpha\gamma}\tilde{\boldsymbol{y}}' &= X_{\alpha\gamma}\boldsymbol{y}' \oplus X_{\beta\gamma}\boldsymbol{y}' \oplus \tilde{X}_{\alpha\gamma}\tilde{\boldsymbol{y}}' \\ &= y'_\alpha z_\gamma \oplus y'_\gamma z_\alpha \oplus y'_\beta z_\gamma \oplus y'_\gamma z_\beta \oplus \tilde{y}'_\alpha \tilde{z}_\gamma \oplus \tilde{y}'_\gamma \tilde{z}_\alpha \\ &= y'_\alpha z_\gamma \oplus y'_\gamma z_\alpha \oplus y'_\beta z_\gamma \oplus y'_\gamma z_\beta \oplus (y'_\alpha \oplus y'_\beta) z_\gamma \oplus y'_\gamma (z_\alpha \oplus z_\beta) \\ &= 0 \end{aligned} \tag{9.62}$$

by relying on Equation 9.55. Finally, we have

$$\begin{aligned} \tilde{L}_{\alpha\alpha}\boldsymbol{y} \oplus \tilde{X}_{\alpha\alpha}\tilde{\boldsymbol{y}}' &= \tilde{L}_{\alpha\alpha}\boldsymbol{y} \oplus \tilde{y}'_\alpha \tilde{z}_\alpha \oplus \tilde{y}'_\alpha \tilde{z}_\alpha \\ &\equiv |\tilde{P}_\alpha \wedge \boldsymbol{y}| \pmod 2 \\ &\equiv |(P_\alpha \oplus P_\beta) \wedge \boldsymbol{y}| \pmod 2 \\ &\equiv |P_\alpha \wedge \boldsymbol{y}| + |P_\beta \wedge \boldsymbol{y}| \pmod 2 \\ &\equiv 0 \pmod 2. \end{aligned} \tag{9.63}$$

Thus,

$$L\boldsymbol{y} \oplus X\boldsymbol{y}' = \mathbf{0} \implies \tilde{L}\boldsymbol{y} \oplus \tilde{X}\tilde{\boldsymbol{y}}' = \mathbf{0} \tag{9.64}$$

which concludes the proof of Theorem 18. □

Theorem 18 can be exploited to design an algorithm equivalent to the TODD algorithm, but which has a lower complexity. Before introducing this algorithm, we demonstrate how the key mechanism utilized by the TODD algorithm to reduce the T-count can be improved. The conditions given by Equations 9.36 and 9.37 of Theorem 17 are sufficient for Equation 9.38 to hold but they are not necessary. The following theorem gives necessary and sufficient conditions for Equation 9.38 to be satisfied.

Theorem 19. *Let P be a parity table of size $n \times m$ and $P' = P \oplus \boldsymbol{z}\boldsymbol{y}^T$ where $\boldsymbol{z}$ and $\boldsymbol{y}$ are vectors of size n and m respectively and such that $|\boldsymbol{y}| \equiv 0 \pmod 2$. And let L and X be matrices and $\boldsymbol{v}$ be a vector, all with rows labelled by $(\alpha\beta)$ such that*

$$L_{\alpha\beta} = P_\alpha \wedge P_\beta \tag{9.65}$$

$$X_{\alpha\beta,\gamma} = z_\alpha \delta_{\beta\gamma} \oplus z_\beta \delta_{\alpha\gamma} \tag{9.66}$$

$$v_{\alpha\beta} = z_\alpha \wedge z_\beta \tag{9.67}$$

for all α, β, γ satisfying $0 \leq \alpha \leq \beta < n$ and $0 \leq \gamma < n$, and where δ is the Kronecker delta defined as follows:

$$\delta_{\alpha\beta} = \begin{cases} 0 & \text{if } \alpha \neq \beta, \\ 1 & \text{if } \alpha = \beta. \end{cases} \tag{9.68}$$

Then, the equality

$$|P'_\alpha \wedge P'_\beta \wedge P'_\gamma| \equiv |P_\alpha \wedge P_\beta \wedge P_\gamma| \pmod 2 \tag{9.69}$$

holds for all $0 \leq \alpha \leq \beta \leq \gamma < n$ if and only if there exists a vector $\boldsymbol{y}'$ and a Boolean b such that $L\boldsymbol{y} \oplus X\boldsymbol{y}' \oplus b\boldsymbol{v} = \mathbf{0}$.

Proof. In the same way as in the proof of Theorem 18, we will use the labels $(\beta\alpha)$ and $(\alpha\beta)$ to refer to the same unique row, for instance $L_{\beta\alpha} = L_{\alpha\beta}$. Analogously to Equation 9.20, we have

$$\begin{aligned} |P'_\alpha \wedge P'_\beta \wedge P'_\gamma| \equiv |P_\alpha \wedge P_\beta \wedge P_\gamma| + z_\gamma |P_\alpha \wedge P_\beta \wedge \boldsymbol{y}| + z_\beta |P_\alpha \wedge P_\gamma \wedge \boldsymbol{y}| + z_\alpha |P_\beta \wedge P_\gamma \wedge \boldsymbol{y}| \\ + z_\beta z_\gamma |P_\alpha \wedge \boldsymbol{y}| + z_\alpha z_\gamma |P_\beta \wedge \boldsymbol{y}| + z_\alpha z_\beta |P_\gamma \wedge \boldsymbol{y}| + z_\alpha z_\beta z_\gamma |\boldsymbol{y}| \pmod 2 \end{aligned} \tag{9.70}$$

for all α, β, γ satisfying $0 \leq \alpha \leq \beta \leq \gamma < n$. Because $|\boldsymbol{y}| \equiv 0 \pmod 2$, we can deduce from Equation 9.70 that Equations 9.69 holds if and only if the following equation is satisfied

$$|[z_\gamma(P_\alpha \wedge P_\beta) \oplus z_\beta(P_\alpha \wedge P_\gamma) \oplus z_\alpha(P_\beta \wedge P_\gamma) \oplus z_\beta z_\gamma P_\alpha \oplus z_\alpha z_\gamma P_\beta \oplus z_\alpha z_\beta P_\gamma] \wedge \boldsymbol{y}| \equiv 0 \pmod 2 \tag{9.71}$$

for all α, β, γ satisfying $0 \leq \alpha \leq \beta \leq \gamma < n$.

Let $\boldsymbol{z}$ be such that $|\boldsymbol{z}| = 1$, and let α be such that $z_\alpha = 1$. Then, for such vector $\boldsymbol{z}$, Equation 9.71 can be rewritten as

$$|P_\beta \wedge P_\gamma \wedge \boldsymbol{y}| \equiv 0 \pmod 2 \tag{9.72}$$

where β, γ are satisfying $\beta \neq \alpha$, $\gamma \neq \alpha$, and as

$$|P_\gamma \wedge \boldsymbol{y}| \equiv 0 \pmod 2 \tag{9.73}$$

where β, γ are satisfying $\beta = \alpha$ and $\gamma \neq \alpha$. Note that Equation 9.71 is necessarily satisfied in the case where $\alpha = \beta = \gamma$, and that if Equation 9.72 is satisfied then Equation 9.73 is also satisfied. Therefore, proving Theorem 19 in the case where $|\boldsymbol{z}| = 1$ can then be done by showing that there exists a vector $\boldsymbol{y'}$ and a Boolean b such that $L\boldsymbol{y} \oplus X\boldsymbol{y'} \oplus b\boldsymbol{v} = \boldsymbol{0}$ if and only if Equation 9.72 is satisfied. By definition we have $X_{\beta\gamma} = \boldsymbol{0}$ and $v_{\beta\gamma} = 0$ for all β, γ satisfying $\beta \neq \alpha$ and $\gamma \neq \alpha$. Hence,

$$L_{\beta\gamma}\boldsymbol{y} \oplus X_{\beta\gamma}\boldsymbol{y'} \oplus bv_{\beta\gamma} = L_{\beta\gamma}\boldsymbol{y} = |P_\beta \wedge P_\gamma \wedge \boldsymbol{y}| \pmod 2 \tag{9.74}$$

for all β, γ satisfying $\beta \neq \alpha$ and $\gamma \neq \alpha$, and so if $L\boldsymbol{y} \oplus X\boldsymbol{y'} \oplus b\boldsymbol{v} = \boldsymbol{0}$ then $L_{\beta\gamma}\boldsymbol{y} = 0$ which imply that Equation 9.72 is satisfied. Conversely, if Equation 9.72 is satisfied then we must have

$$L_{\beta\gamma}\boldsymbol{y} \oplus X_{\beta\gamma}\boldsymbol{y'} \oplus bv_{\beta\gamma} = L_{\beta\gamma}\boldsymbol{y} = 0 \tag{9.75}$$

for all β, γ satisfying $\beta \neq \alpha$ and $\gamma \neq \alpha$ and for any vector $\boldsymbol{y'}$ and Boolean b. Moreover we have

$$L_{\alpha\alpha}\boldsymbol{y} \oplus X_{\alpha\alpha}\boldsymbol{y'} \oplus bv_{\alpha\alpha} = L_{\alpha\alpha}\boldsymbol{y} \oplus b \tag{9.76}$$

because $X_{\alpha\alpha} = \boldsymbol{0}$ and $v_{\alpha\alpha} = 1$. And

$$L_{\alpha\beta}\boldsymbol{y} \oplus X_{\alpha\beta}\boldsymbol{y'} \oplus bv_{\alpha\beta} = L_{\alpha\beta}\boldsymbol{y} \oplus y'_\beta \tag{9.77}$$

where $\beta \neq \alpha$ by using Equation 9.55 and because $v_{\alpha\beta} = 0$. Let $\boldsymbol{y'}$ be such that $y'_\beta = L_{\alpha\beta}\boldsymbol{y}$ for all $\beta \neq \alpha$ and b be such that $b = L_{\alpha\alpha}\boldsymbol{y}$, then we have $L\boldsymbol{y} \oplus X\boldsymbol{y'} \oplus b\boldsymbol{v} = \boldsymbol{0}$. Thus, we proved that

$$L\boldsymbol{y} \oplus X\boldsymbol{y'} \oplus b\boldsymbol{v} = \boldsymbol{0} \iff |P_\beta \wedge P_\gamma \wedge \boldsymbol{y}| \equiv 0 \pmod 2 \tag{9.78}$$

for all β, γ satisfying $\beta \neq \alpha$ and $\gamma \neq \alpha$, and so Theorem 19 is true in the case where $|\boldsymbol{z}| = 1$.

Let B be a full rank binary matrix of size $n \times n$ which represents a change of basis, and let $\tilde{L}$, $\tilde{X}$, $\tilde{\boldsymbol{v}}$ be constructed in the same way as L, X and $\boldsymbol{v}$ but with respect to BP and $B\boldsymbol{z}$. The proof of Theorem 19 can be completed by proving that if there exists a vector $\boldsymbol{y'}$ and a Boolean b such that $L\boldsymbol{y} \oplus X\boldsymbol{y'} \oplus b\boldsymbol{v} = \boldsymbol{0}$ then there exists a vector $\tilde{\boldsymbol{y}}'$ and a Boolean $\tilde{b}$ such that $\tilde{L}\boldsymbol{y} \oplus \tilde{X}\tilde{\boldsymbol{y}}' \oplus \tilde{b}\tilde{\boldsymbol{v}} = \boldsymbol{0}$. Suppose this proposition to be true and let B be such that $|B\boldsymbol{z}| = 1$. Then, as previously demonstrated, Equation 9.69 holds for BP and $B\boldsymbol{z}$ if and only if there exists a vector $\tilde{\boldsymbol{y}}'$ and a Boolean $\tilde{b}$ such that $\tilde{L}\boldsymbol{y} \oplus \tilde{X}\tilde{\boldsymbol{y}}' \oplus \tilde{b}\tilde{\boldsymbol{v}} = \boldsymbol{0}$, which would imply that there exists a vector $\boldsymbol{y'}$ and a Boolean b such that $L\boldsymbol{y} \oplus X\boldsymbol{y'} \oplus b\boldsymbol{v} = \boldsymbol{0}$ if and only if Equation 9.69 holds for P and $\boldsymbol{z}$.

Let $\tilde{\boldsymbol{z}}$ and $\tilde{P}$ be such that $\tilde{z}_\alpha = z_\alpha \oplus z_\beta$ and $\tilde{P}_\alpha = P_\alpha \oplus P_\beta$ for some fixed α, β satisfying $\alpha \neq \beta$ and $\tilde{z}_\gamma = z_\gamma$, $\tilde{P}_\gamma = P_\gamma$ for all γ such that $\gamma \neq \alpha$. Let $\tilde{L}$, $\tilde{X}$ and $\tilde{\boldsymbol{v}}$ be constructed in the same way as L, X and $\boldsymbol{v}$ but with respect to $\tilde{P}$ and $\tilde{\boldsymbol{z}}$. Let's assume that $L\boldsymbol{y} \oplus X\boldsymbol{y'} \oplus b\boldsymbol{v} = \boldsymbol{0}$, we will show that it implies $\tilde{L}\boldsymbol{y} \oplus \tilde{X}\tilde{\boldsymbol{y}}' \oplus b\tilde{\boldsymbol{v}} = \boldsymbol{0}$, where $\tilde{\boldsymbol{y}}'$ satisfies $\tilde{y}'_\alpha = y'_\alpha \oplus y'_\beta$ and $\tilde{y}'_\gamma = y'_\gamma$ for all γ such that $\gamma \neq \alpha$. By using Equation 9.55 we can deduce that

$$\tilde{X}_{\gamma\gamma'}\tilde{\boldsymbol{y}}' = \tilde{y}'_\gamma \tilde{z}_{\gamma'} \oplus \tilde{y}'_{\gamma'} \tilde{z}_\gamma = y'_\gamma z_{\gamma'} \oplus y'_{\gamma'} z_\gamma = X_{\gamma\gamma'}\boldsymbol{y'} \tag{9.79}$$

for all γ, γ' satisfying $\gamma \neq \alpha$ and $\gamma' \neq \alpha$. And from the definitions of the vectors $\tilde{\boldsymbol{v}}$ and $\boldsymbol{v}$, we can deduce that

$$\tilde{v}_{\gamma\gamma'} = \tilde{z}_\gamma \wedge \tilde{z}'_\gamma = z_\gamma \wedge z'_\gamma = v_{\gamma\gamma'} \tag{9.80}$$

for all γ, γ' satisfying $\gamma \neq \alpha$ and $\gamma' \neq \alpha$. Equations 9.79 and 9.80 imply that

$$\tilde{L}_{\gamma\gamma'}\boldsymbol{y} \oplus \tilde{X}_{\gamma\gamma'}\tilde{\boldsymbol{y}}' \oplus b\tilde{v}_{\gamma\gamma'} = L_{\gamma\gamma'}\boldsymbol{y} \oplus X_{\gamma\gamma'}\boldsymbol{y'} \oplus bv_{\gamma\gamma'} = 0 \tag{9.81}$$

for all γ, γ' satisfying $\gamma \neq \alpha$ and $\gamma' \neq \alpha$. Furthermore, for all γ such that $\gamma \neq \alpha$, we have

$$\begin{aligned}
\tilde{L}_{\alpha\gamma}\boldsymbol{y} &\equiv |\tilde{P}_\alpha \wedge \tilde{P}_\gamma \wedge \boldsymbol{y}| \pmod 2 \\
&\equiv |(P_\alpha \oplus P_\beta) \wedge P_\gamma \wedge \boldsymbol{y}| \pmod 2 \\
&\equiv |P_\alpha \wedge P_\gamma \wedge \boldsymbol{y}| + |P_\beta \wedge P_\gamma \wedge \boldsymbol{y}| \pmod 2 \\
&\equiv L_{\alpha\gamma}\boldsymbol{y} + L_{\beta\gamma}\boldsymbol{y} \pmod 2 \\
&\equiv X_{\alpha\gamma}\boldsymbol{y'} + bv_{\alpha\gamma} + X_{\beta\gamma}\boldsymbol{y'} + bv_{\beta\gamma} \pmod 2
\end{aligned} \tag{9.82}$$

which entails

$$\begin{aligned}
&\tilde{L}_{\alpha\gamma}\boldsymbol{y} \oplus \tilde{X}_{\alpha\gamma}\tilde{\boldsymbol{y}}' \oplus b\tilde{v}_{\alpha\gamma} \\
&= X_{\alpha\gamma}\boldsymbol{y}' \oplus X_{\beta\gamma}\boldsymbol{y}' \oplus \tilde{X}_{\alpha\gamma}\tilde{\boldsymbol{y}}' \oplus bv_{\alpha\gamma} \oplus bv_{\beta\gamma} \oplus b\tilde{v}_{\alpha\gamma} \\
&= y'_\alpha z_\gamma \oplus y'_\gamma z_\alpha \oplus y'_\beta z_\gamma \oplus y'_\gamma z_\beta \oplus \tilde{y}'_\alpha \tilde{z}_\gamma \oplus \tilde{y}'_\gamma \tilde{z}_\alpha \oplus b(z_\alpha \wedge z_\gamma) \oplus b(z_\beta \wedge z_\gamma) \oplus b(\tilde{z}_\alpha \wedge z_\gamma) \\
&= y'_\alpha z_\gamma \oplus y'_\gamma z_\alpha \oplus y'_\beta z_\gamma \oplus y'_\gamma z_\beta \oplus (y'_\alpha \oplus y'_\beta) z_\gamma \oplus y'_\gamma (z_\alpha \oplus z_\beta) \\
&\quad \oplus b(z_\alpha \wedge z_\gamma) \oplus b(z_\beta \wedge z_\gamma) \oplus b((z_\alpha \oplus z_\beta) \wedge z_\gamma) \\
&= 0
\end{aligned} \tag{9.83}$$

by relying on Equation 9.55. Finally, we have

$$\begin{aligned}
\tilde{L}_{\alpha\alpha}\boldsymbol{y} \oplus \tilde{X}_{\alpha\alpha}\tilde{\boldsymbol{y}}' \oplus b\tilde{v}_{\alpha\alpha} &= \tilde{L}_{\alpha\alpha'}\boldsymbol{y} \oplus \tilde{y}'_\alpha \tilde{z}_\alpha \oplus \tilde{y}'_\alpha \tilde{z}_\alpha \oplus b\tilde{z}_\alpha \\
&= \tilde{L}_{\alpha\alpha'}\boldsymbol{y} \oplus b(z_\alpha \oplus z_\beta) \\
&\equiv |\tilde{P}_\alpha \wedge \boldsymbol{y}| + bv_{\alpha\alpha} + bv_{\beta\beta} \pmod 2 \\
&\equiv |(P_\alpha \oplus P_\beta) \wedge \boldsymbol{y}| + bv_{\alpha\alpha} + bv_{\beta\beta} \pmod 2 \\
&\equiv |P_\alpha \wedge \boldsymbol{y}| + |P_\beta \wedge \boldsymbol{y}| + bv_{\alpha\alpha} + bv_{\beta\beta} \pmod 2 \\
&= L_{\alpha\alpha} \oplus bv_{\alpha\alpha} \oplus L_{\beta\beta} \oplus bv_{\beta\beta} \\
&= 0.
\end{aligned} \tag{9.84}$$

Thus,

$$L\boldsymbol{y} \oplus X\boldsymbol{y}' \oplus b\boldsymbol{v} = \boldsymbol{0} \implies \tilde{L}\boldsymbol{y} \oplus \tilde{X}\tilde{\boldsymbol{y}}' \oplus b\tilde{v} = \boldsymbol{0} \tag{9.85}$$

which concludes the proof of Theorem 19. □

Consider the algorithm whose pseudo-code is given in Algorithm 14 and which is built upon Theorem 19. The algorithm starts by performing a call to the `TOHPE` algorithm (Algorithm 13). The reason for calling the `TOHPE` algorithm is that it is more efficient to first search for a vector $\boldsymbol{y}$ satisfying $L\boldsymbol{y} = \boldsymbol{0}$ and exploit Theorem 14 than to search for vectors $\boldsymbol{y}$, $\boldsymbol{y}'$ and a Boolean b satisfying $L\boldsymbol{y} \oplus X\boldsymbol{y}' \oplus b\boldsymbol{v} = \boldsymbol{0}$, where $L, X, \boldsymbol{v}$ are defined as in Theorem 19. The algorithm then proceeds by creating the matrix L and the set Z which contains all the vectors $\boldsymbol{z}$ that can potentially be used to transform P via Theorem 19 and reduce its number of columns. For each vector $\boldsymbol{z} \in Z$, the algorithm constructs the set $S^{(\boldsymbol{z})}$ of pairs of indices $\{i, j\}$ satisfying $P_{:,i} \oplus P_{:,j} = \boldsymbol{z}$ in the case where $i \neq j$ or satisfying $P_{:,i} = \boldsymbol{z}$ in the case where $i = j$, as well as the matrix $X^{(\boldsymbol{z})}$ and the vector $\boldsymbol{v}^{(\boldsymbol{z})}$ which are constructed in the same way as in Theorem 19. The set of vectors $\boldsymbol{y}$ satisfying $L\boldsymbol{y} \oplus X^{(\boldsymbol{z})}\boldsymbol{y}' \oplus b\boldsymbol{v}^{(\boldsymbol{z})} = \boldsymbol{0}$ for some vector $\boldsymbol{y}'$ and some Boolean b are stored in $N^{(\boldsymbol{z})}$. The algorithm then computes the pair of vectors $\boldsymbol{z}, \boldsymbol{y}$ where $\boldsymbol{z} \in Z$ and $\boldsymbol{y} \in N^{(\boldsymbol{z})}$ for which the number of duplicated and all-zero columns in P' is maximized, where P' is defined as follows:

$$P' = \begin{cases} P \oplus \boldsymbol{z}\boldsymbol{y}^T & \text{if } |\boldsymbol{y}| \equiv 0 \pmod 2, \\ \begin{bmatrix} P \oplus \boldsymbol{z}\boldsymbol{y}^T & \boldsymbol{z} \end{bmatrix} & \text{otherwise.} \end{cases} \tag{9.86}$$

Algorithm 14: Faster version of the third order duplicate and destroy algorithm

Input: A parity table P of size $n \times m$.

Output: An equivalent parity table with an optimized number of columns.

procedure `FastTODD`(P)
  $P \leftarrow$ `TOHPE`(P)
  $Z \leftarrow \{P_{:,i} \oplus P_{:,j} \mid 0 \leq i < j < m\} \cup \{P_{:,i} \mid 0 \leq i < m\}$
  $L \leftarrow$ matrix whose rows are forming the set $\{P_i \wedge P_j \mid 0 \leq i \leq j < n\}$ labelled by ij
  forall $\boldsymbol{z} \in Z$ **do**
    $S^{(\boldsymbol{z})} \leftarrow \{\{i,j\} \mid P_{:,i} \oplus P_{:,j} = \boldsymbol{z}\} \cup \{\{i,i\} \mid P_{:,i} = \boldsymbol{z}\}$
    $X^{(\boldsymbol{z})} \leftarrow$ matrix such that $X^{(\boldsymbol{z})}_{\alpha\beta,\gamma} = z_\alpha \delta_{\beta\gamma} \oplus z_\beta \delta_{\alpha\gamma}$
    $\boldsymbol{v}^{(\boldsymbol{z})} \leftarrow$ vector such that $v_{\alpha\beta} = z_\alpha \wedge z_\beta$
    $N^{(\boldsymbol{z})} \leftarrow$ generators of the set $\{\boldsymbol{y} \mid L\boldsymbol{y} \oplus X^{(\boldsymbol{z})}\boldsymbol{y}' \oplus b\boldsymbol{v}^{(\boldsymbol{z})} = \boldsymbol{0}\}$
  end
  $\boldsymbol{z}, \boldsymbol{y} \leftarrow \underset{\boldsymbol{z} \in Z, \boldsymbol{y} \in N^{(\boldsymbol{z})}}{\operatorname{argmax}} \Big\{ -|\boldsymbol{y}| \pmod 2 + \sum_{\{i,j\} \in S^{(\boldsymbol{z})}} 2(y_i \oplus y_j) + \delta_{ij}\left[y_i + 2(y_i \oplus 1)(|\boldsymbol{y}| \pmod 2)\right] \Big\}$
  $P' \leftarrow P \oplus \boldsymbol{z}\boldsymbol{y}^T$
  if $|\boldsymbol{y}| \equiv 1 \pmod 2$ **then**
    $P' \leftarrow \begin{bmatrix} P' & \boldsymbol{z} \end{bmatrix}$
  end
  $P' \leftarrow P'$ with all its duplicated and all-zero columns removed
  if size(P') < size(P) **then**
    return `FastTODD`(P')
  end
  return P

This number is given by the following objective function:

$$-|\boldsymbol{y}| \pmod 2 + \sum_{\{i,j\} \in S^{(\boldsymbol{z})}} 2(y_i \oplus y_j) + \delta_{ij}\left[y_i + 2(y_i \oplus 1)(|\boldsymbol{y}| \pmod 2)\right] \tag{9.87}$$

which is the same as the one used in Algorithm 13. After computing the vectors $\boldsymbol{z}$ and $\boldsymbol{y}$ maximizing this function, Algorithm 14 apply the transformation on the parity table and a recursive call is performed if the number of columns in the parity table has successfully been reduced, otherwise the parity table is returned.

Complexity analysis. Performing a call to the `TOHPE` algorithm induces a complexity of $\mathcal{O}(n^2m^3)$. The set Z and the sets $S^{(\boldsymbol{z})}$ can be created with $\mathcal{O}(nm^2)$ operations. We can take advantage of the fact that L doesn't depend on $\boldsymbol{z}$ to compute $N^{(\boldsymbol{z})}$ efficiently. Let $\tilde{L}$ be the reduced column echelon form of L and let B be the matrix such that $\tilde{L}B = L$. Let $\tilde{X}^{(\boldsymbol{z})}$ be such that

$$\tilde{X}^{(\boldsymbol{z})}_{:,\ell} = X_{:,\ell} \bigoplus_{j \in J^{(\ell)}} \tilde{L}_{:,j} \tag{9.88}$$

where $J^{(\ell)}$ is defined as follows:

$$J^{(\ell)} = \{j \mid \exists i \text{ such that } X^{(\boldsymbol{z})}_{i,\ell} = 1, \tilde{L}_{i,j} = 1, \tilde{L}_{i,k} = 0, \forall k \neq j\}. \tag{9.89}$$

And let $\tilde{\boldsymbol{v}}^{(\boldsymbol{z})}$ be such that

$$\tilde{\boldsymbol{v}}^{(\boldsymbol{z})} = \boldsymbol{v} \bigoplus_{j \in J} \tilde{L}_{:,j} \tag{9.90}$$

where J is defined as follows:

$$J = \{j \mid \exists i \text{ such that } v_i = 1, \tilde{L}_{i,j} = 1, \tilde{L}_{i,k} = 0, \forall k \neq j\}. \tag{9.91}$$

If there exists no vector $\boldsymbol{y}$ such that $\boldsymbol{y} \neq \mathbf{0}$ and $L\boldsymbol{y} = \mathbf{0}$ (which is the case after having executed the `TOHPE` algorithm, except potentially for $\boldsymbol{y} = \mathbf{1}$ which can be ignored as it cannot be used to reduce the number of columns in the parity table), then there exists a vector $\boldsymbol{y} \neq \mathbf{0}$, a vector $\boldsymbol{y}'$ and a Boolean b such that $L\boldsymbol{y} \oplus X^{(\boldsymbol{z})}\boldsymbol{y}' \oplus b\boldsymbol{v}^{(\boldsymbol{z})} = \mathbf{0}$ if and only if there exists a vector $\tilde{\boldsymbol{y}}'$ and a bolean $\tilde{b}$ such that $\tilde{X}^{(\boldsymbol{z})}\tilde{\boldsymbol{y}}' \oplus \tilde{b}\tilde{\boldsymbol{v}}^{(\boldsymbol{z})} = \mathbf{0}$. If the equation $\tilde{X}^{(\boldsymbol{z})}\tilde{\boldsymbol{y}}' \oplus \tilde{b}\tilde{\boldsymbol{v}}^{(\boldsymbol{z})} = \mathbf{0}$ is satisfied then the associated vector $\boldsymbol{y}$ is equal to

$$\boldsymbol{y} = \begin{bmatrix} \tilde{B}^{(X)} & \tilde{B}^{(\boldsymbol{v})} \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{y}}' \\ \tilde{b} \end{bmatrix} \tag{9.92}$$

where $\tilde{B}^{(X)}$ is defined as follows:

$$\tilde{B}^{(X)}_{:,\ell} = \bigoplus_{j \in J^{(\ell)}} \tilde{L}_{:,j} \tag{9.93}$$

and $\tilde{B}^{(\boldsymbol{v})}$ is defined as follows:

$$\tilde{B}^{(\boldsymbol{v})} = \bigoplus_{j \in J} \tilde{L}_{:,j}. \tag{9.94}$$

Constructing the matrix $\tilde{X}^{(\boldsymbol{z})}$ can be done in $\mathcal{O}(n^4)$ operations because $X^{(\boldsymbol{z})}$ has n columns and $\mathcal{O}(n^2)$ rows, and $|X^{(\boldsymbol{z})}_{:,\ell}| < n$ which imply that the set $J^{(\ell)}$ contains no more than n elements. Constructing the vector $\tilde{\boldsymbol{v}}^{(\boldsymbol{z})}$ can also be done in $\mathcal{O}(n^4)$ operations because $\boldsymbol{v}^{(\boldsymbol{z})}$ has $\mathcal{O}(n^2)$ rows and the set J contains $\mathcal{O}(n^2)$ elements. Performing a Gaussian elimination on the matrix $\begin{bmatrix} \tilde{X}^{(\boldsymbol{z})} & \tilde{\boldsymbol{v}}^{(\boldsymbol{z})} \end{bmatrix}$ to compute $\tilde{\boldsymbol{y}}'$ and $\tilde{b}$ requires $\mathcal{O}(n^4)$ operations because the matrix has $\mathcal{O}(n)$ columns and $\mathcal{O}(n^2)$ rows. The matrix $\tilde{B}^{(X)}$ and the vector $\tilde{B}^{(\boldsymbol{v})}$ can be computed with the same computational complexity as the matrix $\tilde{X}^{(\boldsymbol{z})}$ and the vector $\tilde{\boldsymbol{v}}^{(\boldsymbol{z})}$. The right nullspace of $\begin{bmatrix} \tilde{X}^{(\boldsymbol{z})} & \tilde{\boldsymbol{v}}^{(\boldsymbol{z})} \end{bmatrix}$ is generated by $\mathcal{O}(n)$ pairs of vector and Boolean $\tilde{\boldsymbol{y}}', \tilde{b}$. For each one of these pairs, the associated vector $\boldsymbol{y} \in N^{(\boldsymbol{z})}$ for which there exists a vector $\boldsymbol{y}'$ and a Boolean b satisfying $L\boldsymbol{y} \oplus X^{(\boldsymbol{z})}\boldsymbol{y}' \oplus b\boldsymbol{v}^{(\boldsymbol{z})} = \mathbf{0}$ can be computed in $\mathcal{O}(n^3)$ by using Equation 9.92, for a total complexity of $\mathcal{O}(n^4)$. This must be done for all the $\mathcal{O}(m^2)$ vectors $\boldsymbol{z}$, which implies a complexity of $\mathcal{O}(n^4m^2)$ for computing all the sets $N^{(\boldsymbol{z})}$. The vectors $\boldsymbol{z}$ and $\boldsymbol{y}$ satisfying the argmax function can be computed in $\mathcal{O}(n^3m^2)$ operations because the union of all the $S^{(\boldsymbol{z})}$ sets contains $\mathcal{O}(m^2)$ elements, computing the value $|\boldsymbol{y}| \pmod 2$ requires $\mathcal{O}(n^2)$ operations and there are $\mathcal{O}(nm^2)$ vectors $\boldsymbol{y}$. Updating the parity table P induces $\mathcal{O}(nm)$ operations. Finally, the algorithm is performing no more than m recursive calls. Thus, the overall worst-case complexity of Algorithm 14 is $\mathcal{O}(n^4m^3)$.

This complexity is significantly better than the $\mathcal{O}(n^3m^5)$ complexity of the original `TODD` algorithm. Note that $n \leq m$, otherwise the instance of the problem can be simplified by eliminating a row of the parity table. Indeed, if $n > m$, then a change of basis can be performed onto the parity table P so that there exists a row α satisfying $P_\alpha = \mathbf{0}$, this row can then be removed from the parity table P which leads to a smaller instance of the problem.

9.3 Benchmarks

In this section we evaluate the performances of the algorithms we presented, and we compare them to state-of-the-art alternatives. The benchmarks are performed on a set of circuits commonly used to compare the performances of T-count optimizers, which were obtained from References [127] and [128]. We also include a circuit implementing the block cipher DEFAULT, as given in Reference [129].

In Subsection 9.3.1, we benchmark the T-count optimizers presented in Section 9.2. We consider the case where ancillary qubits can be used, as well as the case where no ancillary qubits are used. Then, in Subsection 9.3.2, we compare the performance of our algorithms with state-of-the-art methods for the optimization of multiplication circuits in finite fields.

9.3.1 `TOHPE` and `FastTODD` algorithms

In this subsection we compare the performances of the `TOHPE` procedure (Algorithm 13), the `FastTODD` procedure (Algorithm 14), the `PHAGE` procedure of Reference [112], the `TODD` procedure of Reference [108] and the deep reinforcement learning method of Reference [131] named AlphaTensor-Quantum. The `PHAGE` procedure was implemented in Haskell by the authors of the method [130], while the `TOHPE`, `FastTODD` and `TODD` procedures were implemented in the Rust programming language. Our implementations of the `TOHPE` and `FastTODD` procedures used for the benchmarks are open source [8].

The execution times of AlphaTensor-Quantum are not reported in the benchmarks because they were not provided in Reference [131]. However, due to the nature of the method, we can expect it to have much longer execution times than all the other methods. Also, some circuits had to be split into several parts for optimizing them with AlphaTensor-Quantum in a reasonable time.

In Reference [131], it is proposed to optimize the number of T gates by implementing CCZ gates using a gadgetization technique presented in Reference [163]. This gadgetization technique uses a ancillary qubit to implement the CCZ with 4 T gates instead of 7 T gates in the case where no ancillary qubits are used. Using this gadgetization technique can therefore lead to a better T-count. However, if the purpose of optimizing the number of T gates is to use fewer resources to implement them using magic state distillation protocols, then this approach can be counterproductive. Indeed, as shown in Reference [157], an implementation of a phase polynomial associated with a third order homogeneous weighted polynomial (as in Equation 9.21) can benefit from a free round of distillation with quadratic error suppression. By using an ancillary qubit, the gadgetization technique for implementing the CCZ gate breaks this important property of the weighted polynomial. Such protocols for implementing and distilling a phase polynomial in a single step are refered to as synthillation protocols. The simplest example of this is that the CCZ gate can be implemented more efficiently by using the corresponding synthillation protocol rather than by distilling 4 magic states and then implementing the CCZ gate using the gadgetization technique. Concretely, for distilling n CCZ gates with quadratic error suppression, the synthillation protocol consumes $\mathcal{O}(6n)$ magic states, whereas using the gadgetization technique would consume $\mathcal{O}(12n)$ magic states by using the Bravyi-Haah protocol (which is the magic state distillation protocol with quadratic error suppression that consumes the fewest number of magic states, up to a small constant) [164]. That is why, in order to maintain consistent metrics and to assign the same cost to each T gate, we won't consider the gadgetization technique of Reference [163] for implementing the CCZ gates in the benchmarks.

Also, the only gate count metric considered in the benchmarks will be the T-count. It

	Pre-optimization			`Alpha` [131]	`PHAGE` [112]		`TOHPE`		`FastTODD`		`TODD` [108]	
Circuit	n	h	T-count	T-count	T-count	t (s)	T-count	t (s)	T-count	t (s)	T-count	t (s)
Adder_8	24	37	173	139	173	25	**119**	0	119	1	128	5580
Barenco Tof_3	5	3	16	13	13	0	13	0	13	0	14	0
Barenco Tof_4	7	7	28	23	26	1	23	0	23	0	24	0
Barenco Tof_5	9	11	40	33	39	11	33	0	33	0	34	0
Barenco Tof_{10}	19	31	100	83	100	27	83	0	83	0	84	237
CSLA MUX_3	15	6	62	39	46	39	39	0	39	0	42	6
CSUM MUX_9	30	12	84	71	84	28	71	0	71	0	72	87
DEFAULT	640	11936	39744	-	-	-	**38638**	-	38638	-	39744	-
Grover_5	9	68	166	152	166	26	**143**	0	143	3	144	6749
Ham_{15} (high)	20	331	1019	773	1019	37	691	2	**643**	-	1001	-
Ham_{15} (low)	17	29	97	**73**	97	26	77	0	77	0	76	453
Ham_{15} (med)	17	54	212	156	212	28	147	0	**137**	19	148	21472
HWB_6	7	20	75	51	68	117	51	0	51	0	51	29
HWB_8	12	1103	3517	-	3517	228	**2763**	169	2763	-	3517	-
HWB_{10}	16	5191	15891	-	15891	10422	**12845**	51781	12845	-	15891	-
HWB_{11}	15	14441	44500	-	-	-	**42643**	-	42643	-	44500	-
Mod Adder_{1024}	28	304	1011	762	1010	36	657	2	**575**	-	957	-
Mod Mult_{55}	9	3	35	17	21	0	17	0	17	0	18	0
Mod Red_{21}	11	17	73	51	72	25	51	0	51	0	53	21
Mod5_4	5	0	8	7	7	0	7	0	7	0	8	0
QCLA Adder_{10}	36	25	162	135	159	26	113	0	**109**	4	111	4216
QCLA Com_7	24	18	95	59	91	28	59	0	59	0	60	149
QCLA Mod_7	26	58	237	199	237	26	167	0	**159**	41	168	60415
QFT_4	5	38	67	53	55	33	53	0	53	0	55	54
RC Adder_6	14	10	47	37	44	32	37	0	37	0	38	2
Tof_3	5	2	15	13	13	0	13	0	13	0	14	0
Tof_4	7	4	23	19	20	0	19	0	19	0	20	0
Tof_5	9	6	31	25	28	1	25	0	25	0	26	0
Tof_{10}	19	16	71	55	69	26	55	0	55	0	56	24
VBE Adder_3	10	4	24	19	22	1	19	0	19	0	20	0

Table 9.1: Comparison of different procedures for the optimization of the number of T gates. n represents to the number of input qubits in the circuits. h denotes the number of internal Hadamard gates in the input circuits. All internal Hadamard gates were gadgetized, resulting in h ancillary qubits. The T-count after optimization and the execution time is reported for each procedure. A blank entry in the execution time indicates that the execution couldn't be carried out in less than a day. In such cases, the reported T-count is the one obtained after 24 hours of execution. A blank entry for the AlphaTensor-Quantum column indicates that no data were reported by the authors on the corresponding circuit. A blank entry in the T-count for the `PHAGE` procedure indicates that the execution was halted because the memory usage exceeded 8 terabytes.

is important to note that all the methods considered in the benchmarks for optimization of the number of T gates can significantly increase the number of CNOT gates in the circuit. If necessary, a phase polynomial synthesis algorithm can be used to perform the synthesis of the phase polynomial produced by the T-count optimizer with an optimized number of CNOT gates [7], [122].

We first consider the case where internal Hadamard gates are gadgetized in in Subsection 9.3.1.1, and then the case where no ancillary qubits are used in Subsection 9.3.1.2.

9.3.1.1 With ancillary qubits

We compare the performances of the algorithms in the case where all the internal Hadamard gates have been gadgetized, as explained in Section 9.1.1. All the circuits were pre-optimized using the `FastTMerge` procedure (Algorithm 11) presented in Chapter 7, followed by the `InternalHOpt` procedure (Algorithm 8), presented in Chapter 6, to minimize the number of internal Hadamard gates. This pre-optimization is important as reducing the number of internal Hadamard gates leads to fewer ancillary qubits when these Hadamard gates are gadgetized, and therefore improve the efficiency of the T-count optimizers.

The benchmark results are presented in Table 9.1. As expected from the complexity analysis of the algorithms, we can notice that the execution times of the `FastTODD` procedure are much lower than the execution times of the `TODD` procedure. Moreover, the `FastTODD` procedure is providing the best-known T-count on every circuits, except for the "Ham_{15} (low)" circuit. While the `TODD` procedure and the `FastTODD` procedure are based on the same mechanism for reducing the T-count, the modifications that we made in the `FastTODD` procedure are resulting in significantly better T-count reduction for some circuits. For instance, for the "Mod Adder_{1024}" circuit, the `FastTODD` procedure generates a circuit with a T-count that is 40% lower than the T-count of the circuit generated by the `TODD` procedure. This 40% improvement remains consistent even if we allow the algorithms to terminate by not stopping them after 24 hours.

The `TOHPE` procedure is also achieving an equivalent or better T-count than the state-of-the-art algorithms for all circuits except the "Ham_{15} (low)" circuit. Moreover, on every circuit, the execution times of the `TOHPE` and `FastTODD` procedures are lower than the execution times of the other procedure (for the "HWB_8" and "HWB_{10}" circuits, the `PHAGE` procedure have lower execution times but but does not improve the T-count at all for these circuits). This demonstrates that our algorithms are faster than state-of-the-art T-count optimizers and are producing circuits with a lower or equivalent T-count.

9.3.1.2 Without ancillary qubits

The benchmarks results in the case where the internal Hadamard gates are not gadgetized are presented in Table 9.2. All the circuits were pre-optimized using the `FastTMerge` procedure. Then the circuits were divided into Hadamard-free subcircuits interposed with Clifford circuits, as described in Subsection 9.1.1 and using Algorithm 12 to minimize the number of Hadamard-free subcircuits.

Similarly to the case where the internal Hadamard gates are gadgetized, the `FastTODD` procedure provides the best-known T-count for all the circuits. Also, the `TOHPE` and `FastTODD` procedures have the lowest execution times. However, we can notice that the number of T gates was not reduced at all for multiple quantum circuits, and the T-count reduction achieved for the other circuits was not very substantial. This indicates the importance of the gadgetization of internal Hadamard gates for lowering the number of T gates once the `FastTMerge` procedure described has been applied. A possible direction for future research work would be to develop effective strategies for optimizing the number of T gates by exploiting the gadgetization of internal Hadamard gates in cases where the number of ancillary at disposal qubits is limited.

9.3.2 Galois field multiplier circuits

In this subsection, we compare the performances of AlphaTensor-Quantum [131], the `FastTODD` procedure, and the synthesis algorithm presented in Chapter 5 for optimizing the number of CCZ and T gates in $\text{GF}(2^n)$ multiplier circuits. The circuits were pre-optimized using the

	Pre-optimization		PHAGE [112]		TOHPE		FastTODD		TODD [108]	
Circuit	n	T-count	T-count	t (s)	T-count	t (s)	T-count	t (s)	T-count	t (s)
Adder_8	24	173	172	17	173	0	**170**	0	172	3
Barenco Tof_3	5	16	16	0	16	0	16	0	16	0
Barenco Tof_4	7	28	28	0	28	0	28	0	28	0
Barenco Tof_5	9	40	40	0	40	0	40	0	40	0
Barenco Tof_{10}	19	100	100	3	100	0	100	0	100	0
CSLA MUX_3	15	62	50	2	**49**	0	49	0	50	0
CSUM MUX_9	30	84	84	79	**73**	0	73	0	76	7
DEFAULT	640	39744	-	-	**39666**	-	39666	-	39744	-
Grover_5	9	166	166	0	166	0	166	0	166	0
Ham_{15} (high)	20	1019	1019	5	1019	0	1019	0	1019	1
Ham_{15} (low)	17	97	94	0	94	0	94	0	94	0
Ham_{15} (med)	17	212	212	3	212	0	212	0	212	1
HWB_6	7	75	75	0	75	0	75	0	75	0
HWB_8	12	3517	3508	32	3498	0	**3487**	0	3493	2
HWB_{10}	16	15891	15858	392	15741	1	**15693**	2	15698	14
HWB_{11}	15	44500	44479	1418	44336	3	**44188**	5	44191	28
HWB_{12}	20	85611	85588	8153	85362	10	**85233**	16	85239	135
Mod Adder_{1024}	28	1011	1010	5636	**1009**	0	1009	0	1010	13
Mod Mult_{55}	9	35	28	0	28	0	28	0	28	0
Mod Red_{21}	11	73	73	0	73	0	73	0	73	0
QCLA Adder_{10}	36	162	161	158	161	0	161	0	161	53
QCLA Com_7	24	95	95	16	95	0	95	0	95	2
QCLA Mod_7	26	237	237	64	237	0	237	0	237	7
QFT_4	5	67	67	0	67	0	66	0	66	0
RC Adder_6	14	47	47	0	47	0	47	0	47	0
Tof_3	5	15	15	0	15	0	15	0	15	0
Tof_4	7	23	23	0	23	0	23	0	23	0
Tof_5	9	31	31	0	31	0	31	0	31	0
Tof_{10}	19	71	71	6	71	0	71	0	71	0
VBE Adder_3	10	24	24	0	24	0	24	0	24	0

Table 9.2: Comparison of different procedures for the optimization of the number of T gates without using any ancillary qubits. The value n corresponds to the number of qubits in the circuits. The T-count after optimization and the execution time is reported for each procedure. A blank entry in the execution time indicates that the execution couldn't be carried out in less than a day. In such cases, the reported T-count is the one obtained after 24 hours of execution. A blank entry in the T-count for the `PHAGE` procedure indicates that the execution was halted because the memory usage exceeded 8 terabytes.

`FastTMerge` procedure, followed by the `InternalHOpt` procedure to minimize the number of internal Hadamard gates. After this optimization, the number of internal Hadamard gates is reduced to zero, thus eliminating the need for Hadamard gate gadgetization and ancillary qubits.

The benchmarks results are presented in Table 9.3. We can notice that AlphaTensor-Quantum finds the best number of CCZ gates in the case where n is equal to 5, 7 and 9. This demonstrates that, while the method of Chapter 5 generates circuits with a subquadratic number of CCZ gates, the optimized circuit may not have an optimal number of CCZ gates (at least in the case where n is not a power of 2). The major drawback of AlphaTensor-Quantum is its large complexity, which imply that it can only be used on small circuits. Conversely, the `FastTODD` algorithms was able to optimize the number of T gates even on the largest circuit, although it did not terminate within 24 hours.

		Alpha [131]		Chap. 5	FastTODD		Chap. 5 & FastTODD	
Circuit	T-count	T-count	CCZ	CCZ	T-count	t (s)	T-count	t (s)
GF(2^2) Mult	18	17	3	3	18	0	17	0
GF(2^3) Mult	45	29	6	6	36	0	**23**	0
GF(2^4) Mult	68	**39**	9	9	49	0	43	0
GF(2^5) Mult	115	**59**	**13**	14	81	0	61	0
GF(2^6) Mult	150	**77**	18	18	113	0	83	0
GF(2^7) Mult	217	**104**	**22**	23	155	1	111	0
GF(2^8) Mult	264	**123**	29	**27**	205	1	135	0
GF(2^9) Mult	351	**161**	**35**	38	257	5	185	2
GF(2^{10}) Mult	410	**196**	46	**42**	315	17	207	3
GF(2^{16}) Mult	1040	-	-	**81**	797	320	**425**	40
GF(2^{32}) Mult	4128	-	-	**243**	3213	49386	**1255**	16485
GF(2^{64}) Mult	16448	-	-	**729**	12503	-	**3817**	-
GF(2^{128}) Mult	65664	-	-	**2187**	50887	-	**11545**	-
GF(2^{256}) Mult	262400	-	-	**6561**	220442	-	**34731**	-
GF(2^{512}) Mult	1048576	-	-	**19683**	1036270	-	**91931**	-

Table 9.3: Comparison of different procedures for the optimization of the number of T and CCZ gates in GF(2^n) multiplier circuits. The CCZ columns refers to the number of CCZ (or Toffoli) gates in the optimized circuit. A blank entry in the execution time for the FastTODD procedure indicates that the execution couldn't be carried out in less than a day. In such cases, the reported T-count is the one obtained after 24 hours of execution. A blank entry for the AlphaTensor-Quantum column indicates that no data were reported by the authors on the corresponding circuit.

The benchmarks results are a good demonstration of the advantages and disadvantages of the AlphaTensor-Quantum method and the FastTODD procedure. Moreover, the results indicate that these two algorithms are not really in direct competition but rather represent complementary approaches. One of the differences between the two methods is that AlphaTensor-Quantum performs a complete resynthesis of the Hadamard-free parts of the quantum circuit. This implies that AlphaTensor-Quantum cannot be used to improve upon existing quantum circuits that may be already well optimized. For instance, whether or not the number of the number of T gates in the "GF(2^n) Mult" input circuit is well optimized will not change the circuit produced by AlphaTensor-Quantum. The main advantage of this resynthesis approach of AlphaTensor-Quantum is that it can be used to discover radically different circuits that have much lower number of T gates. We can clearly see that this is the case for the GF(2^n) multiplier circuits: the input circuit contains $\mathcal{O}(n^2)$ but AlphaTensor-Quantum finds a circuit with a number of CCZ gates similar to the method of Chapter 5 which produces a circuit with $\mathcal{O}(n^{1.58})$ CCZ gates. This demonstrates that AlphaTensor-Quantum could be used to give valuable insights for the design of efficient synthesis methods. On the contrary, the FastTODD procedure does not perform well on the "GF(2^n) Mult" circuits containing $\mathcal{O}(n^2)$ CCZ gates. However, the FastTODD procedure can be used to optimize the number of T gates in the circuit produced by the method given in Chapter 5. The FastTODD procedure could also be used to improve the solution of AlphaTensor-Quantum. For instance, as shown in Table 9.3, some of the circuits

produced by AlphaTensor-Quantum contain an even number of T gates. In such cases, the parity table associated with the circuit satisfies all the conditions of Theorem 14. Consequently, applying the `FastTODD` procedure on these circuits will necessarily reduce the number of T gates.

9.4 Extension to higher levels of the Clifford hierarchy

In this section, we extend the results of Section 9.2 for the optimization of the number of $R_Z(\pi/2^d)$ gates in $\{\text{CNOT}, R_Z(\pi/2^d), R_Z(2\pi/2^d)\}$ circuits, for any non-negative integer d. These gates appear in various quantum algorithms, such as Shor's algorithm [15], and there exists distillation protocols for implementing them fault tolerantly [165,166]. Optimizing the number of $R_Z(\pi/2^d)$ gates may increase the number of $R_Z(2\pi/2^d)$ gates. However, this is motivated by the fact that the $R_Z(\pi/2^d)$ gate is typically more costly to implement than the $R_Z(2\pi/2^d)$ gate.

We formalize the problem of $R_Z(\pi/2^d)$ gates optimization in $\{\text{CNOT}, R_Z(\pi/2^d), R_Z(2\pi/2^d)\}$ circuits in Subsection 9.4.1. Then, in Subsection 9.4.2, we extend Theorem 16 to give an upper bound on the number of $R_Z(\pi/2^d)$ gates in $\{\text{CNOT}, R_Z(\pi/2^d), R_Z(2\pi/2^d)\}$ circuits, which can be satisfied in polynomial time by generalizing Algorithm 13. Finally, in Subsection 9.4.3, we show how we can construct algorithms that are similar to Algorithm 14 for optimizing the number of $R_Z(\pi/2^d)$ gates by providing a generalization of Theorem 19.

9.4.1 Symmetric tensor rank decomposition problem

In this subsection we formalize the problem of $R_Z(\pi/2^d)$ gates optimization in $\{\text{CNOT}, R_Z(\pi/2^d), R_Z(2\pi/2^d)\}$ circuits for any non-negative integer d, by showing its equivalence to the following symmetric tensor rank decomposition problem.

Problem 5 (d-STR)**.** *Let $\mathcal{A} \in \mathbb{Z}_2^{(n,\ldots,n)}$ be a symmetric tensor of order d such that*

$$\mathcal{A}_{\alpha_1,\ldots,\alpha_d} = \mathcal{A}_{\beta_1,\ldots,\beta_d} \tag{9.95}$$

for all $\alpha_1,\ldots,\alpha_d$ and $\beta_1,\ldots,\beta_d$ such that the set equality $\{\alpha_i,\ldots,\alpha_d\} = \{\beta_i,\ldots,\beta_d\}$ is satisfied. Find a Boolean matrix P of size $n \times m$ such that

$$\mathcal{A}_{\alpha_1,\ldots,\alpha_d} = \Big|\bigwedge_{i=1}^{d} P_{\alpha_i}\Big| \pmod 2 \tag{9.96}$$

for all $\alpha_1,\ldots,\alpha_d$ satisfying $0 \leq \alpha_1 \leq \ldots \leq \alpha_d < n$, with minimal m.

We now prove the equivalence between the problem of minimizing the number of $R_Z(\pi/2^d)$ gates in a $\{\text{CNOT}, R_Z(\pi/2^d), R_Z(2\pi/2^d)\}$ circuit for any non-negative integer d and the $(d+1)$-STR problem. The following theorem was first proven in Reference [106] by showing the equivalence between the problem of optimizing $R_Z(\pi/2^d)$ gates and the minimum distance decoding problem for the punctured Reed-Muller code of order $n-d-2$ and length 2^n-1, which is equivalent to the d-STR problem [120].

Theorem 20. *Let d be a non-negative integer. The $R_Z(\pi/2^d)$-count optimization problem over the $\{\text{CNOT}, R_Z(\pi/2^d), R_Z(2\pi/2^d)\}$ gate set and the $(d+1)$-STR problem are equivalent.*

Proof. The CNOT and $R_Z(\theta)$ gates are acting as follows on basis states $|x_1,x_2\rangle$, $|x_1\rangle$:

$$\text{CNOT}|x_1,x_2\rangle = |x_1, x_1 \oplus x_2\rangle \tag{9.97}$$

$$R_Z(\theta)|x_1\rangle = e^{i\theta x_1}|x_1\rangle \tag{9.98}$$

Therefore, the action of an n-qubits $\{\text{CNOT}, R_Z(\pi/2^d), R_Z(2\pi/2^d)\}$ circuit on a basis state $|\boldsymbol{x}\rangle$ has the form

$$|\boldsymbol{x}\rangle \mapsto e^{i\frac{\pi}{2^d}p(\boldsymbol{x})}|g(\boldsymbol{x})\rangle \tag{9.99}$$

where $g : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$ is a linear reversible Boolean function which can be implemented using only CNOT gates, and p is a linear combination of linear Boolean functions:

$$p(\boldsymbol{x}) = \sum_{j=1}^{m} a_j(y_1^{(j)}x_1 \oplus \ldots \oplus y_n^{(j)}x_n) \pmod{2^{d+1}} \tag{9.100}$$

where $\boldsymbol{y}^{(j)} \in \mathbb{Z}_2^n \setminus \{\boldsymbol{0}\}$, $\boldsymbol{a} \in \mathbb{Z}_{2^\ell}^n$ and $m \geq 0$. The function p is called a phase polynomial, we will refer to the Boolean vectors $\boldsymbol{y}^{(j)}$ as the parities of the phase polynomial p and we will refer to $\boldsymbol{a}$ as the weights of the phase polynomial. Notice that if a weight a_j associated to a parity $\boldsymbol{y}^{(j)}$ is odd, then the associated rotation can be implemented using only the $R_Z(2\pi/2^d)$ gate. Therefore, the number of $R_Z(\pi/2^d)$ gates required to implement p is equal to $|\boldsymbol{a} \pmod 2|$. That is why the problem of minimizing the number of $R_Z(\pi/2^d)$ gates in a $\{\text{CNOT}, R_Z(\pi/2^d), R_Z(2\pi/2^d)\}$ circuit consists in finding a phase polynomial p' equivalent to p but with weights $\boldsymbol{a}'$ such that $|\boldsymbol{a}' \pmod 2|$ is minimal. We now prove the following equality, which will be useful to characterize the set of phase polynomials that are equivalent:

$$x_1 \oplus \ldots \oplus x_n = \sum_{k=1}^{n} \sum_{\alpha_1 < \ldots < \alpha_k} (-2)^{k-1} \prod_{i=1}^{k} x_{\alpha_i} \tag{9.101}$$

for all $n \geq 1$ and where $\boldsymbol{x} \in \mathbb{Z}_2^n$. This equality is trivially true for $n = 1$, and for $n = 2$ we can easily verify by case distinction that

$$x_1 \oplus x_2 = x_1 + x_2 - 2x_1x_2. \tag{9.102}$$

Let assume that the equality is true for n and let $\tilde{x}_1 = x_1 \oplus x_{n+1}$, then we have

$$\begin{aligned}
x_1 \oplus \ldots \oplus x_{n+1} &= \tilde{x}_1 \oplus \ldots \oplus x_n \\
&= \tilde{x}_1 \left(1 + \sum_{k=2}^{n} \sum_{1 < \alpha_1 < \ldots < \alpha_{k-1}} (-2)^{k-1} \prod_{i=2}^{k} x_{\alpha_i}\right) \\
&\quad + \sum_{k=1}^{n} \sum_{1 < \alpha_1 < \ldots < \alpha_k} (-2)^{k-1} \prod_{i=1}^{k} x_{\alpha_i} \\
&= (x_1 \oplus x_{n+1}) \left(1 + \sum_{k=2}^{n} \sum_{1 < \alpha_1 < \ldots < \alpha_{k-1}} (-2)^{k-1} \prod_{i=2}^{k} x_{\alpha_i}\right) \\
&\quad + \sum_{k=1}^{n} \sum_{1 < \alpha_1 < \ldots < \alpha_k} (-2)^{k-1} \prod_{i=1}^{k} x_{\alpha_i} \\
&= (x_1 + x_{n+1} - 2x_1x_{n+1}) \left(1 + \sum_{k=2}^{n} \sum_{1 < \alpha_1 < \ldots < \alpha_{k-1}} (-2)^{k-1} \prod_{i=2}^{k} x_{\alpha_i}\right) \\
&\quad + \sum_{k=1}^{n} \sum_{1 < \alpha_1 < \ldots < \alpha_k} (-2)^{k-1} \prod_{i=1}^{k} x_{\alpha_i} \\
&= \sum_{k=1}^{n+1} \sum_{\alpha_1 < \ldots < \alpha_k} (-2)^{k-1} \prod_{i=1}^{k} x_{\alpha_i}
\end{aligned} \tag{9.103}$$

Moreover, the equality holds under the modulo of any even number, and so

$$\begin{aligned} x_1 \oplus \ldots \oplus x_n &= \sum_{k=1}^{n} \sum_{\alpha_1<\ldots<\alpha_k} (-2)^{k-1} \prod_{i=1}^{k} x_{\alpha_i} \pmod{2^{d+1}} \\ &= \sum_{k=1}^{d+1} \sum_{\alpha_1<\ldots<\alpha_k} (-2)^{k-1} \prod_{i=1}^{k} x_{\alpha_i} \pmod{2^{d+1}} \end{aligned} \tag{9.104}$$

where d is a non-negative integer satisfying $d+1 \leq n$. Then we have

$$\begin{aligned} p(\boldsymbol{x}) &= \sum_{j=1}^{m} a_j (y_1^{(j)} x_1 \oplus \ldots \oplus y_n^{(j)} x_n) \pmod{2^{d+1}} \\ &= \sum_{j=1}^{m} a_j \sum_{k=1}^{d+1} \sum_{\alpha_1<\ldots<\alpha_k} (-2)^{k-1} \prod_{i=1}^{k} y_{\alpha_i}^{(j)} x_{\alpha_i} \pmod{2^{d+1}} \\ &= \sum_{k=1}^{d} \sum_{\alpha_1<\ldots<\alpha_k} (-2)^{k-1} c_{\alpha_1,\ldots,\alpha_k} \prod_{i=1}^{k} x_{\alpha_i} \pmod{2^{d+1}} \end{aligned} \tag{9.105}$$

where $c_{\alpha_1,\ldots,\alpha_k} = \sum_{j=1}^{m} a_j \prod_{i=1}^{k} y_{\alpha_1}^{(j)} \pmod{2^{d-k+1}}$. Notice that two phase polynomials p and p' with parities and weights $\boldsymbol{y}^{(j)}$, $\boldsymbol{a}$ and $\boldsymbol{y}'^{(j)}$, $\boldsymbol{a}'$ respectively are equal if and only if

$$\begin{aligned} c_{\alpha_1,\ldots,\alpha_k} &= \sum_{j=1}^{m} a_j \prod_{i=1}^{k} y_{\alpha_1}^{(j)} \pmod{2^{d-k+1}} \\ &= \sum_{j=1}^{m'} a'_j \prod_{i=1}^{k} y'^{(j)}_{\alpha_1} \pmod{2^{d-k+1}} \\ &= c'_{\alpha_1,\ldots,\alpha_k} \end{aligned} \tag{9.106}$$

for all $\alpha_1, \ldots, \alpha_k$ satisfying $\alpha_1 < \ldots < \alpha_k$. Moreover, p and p' are equivalent up to an operator implementable over the $\{\text{CNOT}, R_Z(2\pi/2^d)\}$ gate set if and only if

$$c_{\alpha_1,\ldots,\alpha_k} \equiv c'_{\alpha_1,\ldots,\alpha_k} \pmod{2} \tag{9.107}$$

for all $\alpha_1, \ldots, \alpha_k$ satisfying $\alpha_1 < \ldots < \alpha_k$. Let P and P' be the parity tables (constructed as explained in Section 9.1) associated with p and p' and such that the columns of P and P' are encoding the parities $\boldsymbol{y}^{(j)}, \boldsymbol{y}'^{(j)}$ associated with an odd weight a_j or a'_j. Then we have

$$\begin{aligned} c_{\alpha_1,\ldots,\alpha_k} &\equiv \sum_{j=1}^{m} a_j \prod_{i=1}^{k} y_{\alpha_1}^{(j)} \pmod{2} \\ &\equiv \Big| \bigwedge_{i=1}^{k} P_{\alpha_i} \Big| \pmod{2} \end{aligned} \tag{9.108}$$

for all $\alpha_1, \ldots, \alpha_k$ satisfying $\alpha_1 < \ldots < \alpha_k$ and k satisfying $1 \leq k \leq d+1$. And so p and p' are equivalent up to an operator implementable over the $\{\text{CNOT}, R_Z(2\pi/2^d)\}$ gate set if and only if

$$\Big| \bigwedge_{i=1}^{k} P_{\alpha_i} \Big| \equiv \Big| \bigwedge_{i=1}^{k} P'_{\alpha_i} \Big| \pmod{2} \tag{9.109}$$

for all $\alpha_1, \ldots, \alpha_k$ satisfying $\alpha_1 < \ldots < \alpha_k$ and k satisfying $1 \leq k \leq d+1$. Let $\mathcal{A} \in \mathbb{Z}_2^{(n,\ldots,n)}$ be a symmetric tensor of order $d+1$ such that

$$\mathcal{A}_{\alpha_1,\ldots,\alpha_{d+1}} = |\bigwedge_{i=1}^{d+1} P_{\alpha_i}| \pmod 2 \tag{9.110}$$

for all $\alpha_1, \ldots, \alpha_{d+1}$ satisfying $0 \leq \alpha_1 \leq \ldots \leq \alpha_{d+1} < n$. Then p' is equivalent to p up to an operator implementable over the $\{\text{CNOT}, R_Z(2\pi/2^d)\}$ gate set if and only if

$$\mathcal{A}_{\alpha_1,\ldots,\alpha_{d+1}} = |\bigwedge_{i=1}^{k} P'_{\alpha_i}| \pmod 2 \tag{9.111}$$

for all $\alpha_1, \ldots, \alpha_{d+1}$ satisfying $0 \leq \alpha_1 \leq \ldots \leq \alpha_{d+1} < n$. The number of $R_Z(\pi/2^d)$ gates required to implement p' over the $\{\text{CNOT}, R_Z(\pi/2^d), R_Z(2\pi/2^d)\}$ gate set is equal to the number of column of P'. Thus, minimizing the number of $R_Z(\pi/2^d)$ gates consists in finding a parity table P' satisfying Equation 9.111 with a minimal number of columns, which correponds to the $(d+1)$-STR problem. □

9.4.2 $R_Z(\pi/2^d)$-count upper bound in $\{\text{CNOT}, R_Z(\pi/2^d), R_Z(2\pi/2^d)\}$ circuits

In this section we present an upper bound for the number of $R_Z(\pi/2^d)$ gates in a $\{\text{CNOT}, R_Z(\pi/2^d), R_Z(2\pi/2^d)\}$ circuit, and we present a method for satisfying this upper bound in polynomial time. We first prove the following theorem, which is a generalization of Theorem 14.

Theorem 21. *Let d be a non-negative integer, let P be a parity table of size $n \times m$ and let $P' = P \oplus \boldsymbol{z}\boldsymbol{y}^T$ where $\boldsymbol{z}$ and $\boldsymbol{y}$ are vectors of size n and m respectively such that*

$$|\boldsymbol{y}| \equiv 0 \pmod 2 \tag{9.112}$$

$$|\bigwedge_{i=1}^{d} P_{\alpha_i} \wedge \boldsymbol{y}| \equiv 0 \pmod 2 \tag{9.113}$$

for all $0 \leq \alpha_1 \leq \ldots \leq \alpha_d < n$. Then we have

$$|\bigwedge_{i=1}^{d+1} P'_{\alpha_i}| \equiv |\bigwedge_{i=1}^{d+1} P_{\alpha_i}| \pmod 2 \tag{9.114}$$

for all $0 \leq \alpha_1 \leq \ldots \leq \alpha_{d+1} < n$.

Proof. For all $\alpha_1, \ldots, \alpha_{d+1}$ satisfying $0 \leq \alpha_1 \leq \ldots \leq \alpha_{d+1} < n$, we have:

$$\begin{aligned} |\bigwedge_{i=1}^{d+1} P'_{\alpha_i}| &= |\bigwedge_{i=1}^{d+1} (P_{\alpha_i} \oplus z_{\alpha_i}\boldsymbol{y})| \\ &= |\bigoplus_{k_{d+1}=0}^{1} \ldots \bigoplus_{k_1=0}^{1} \left[\bigwedge_{i=1}^{d+1} (k_i P_{\alpha_i} \vee (1-k_i) z_{\alpha_i}\boldsymbol{y})\right]| \\ &\equiv \sum_{k_{d+1}=0}^{1} \ldots \sum_{k_1=0}^{1} |\left[\bigwedge_{i=1}^{d+1} (k_i P_{\alpha_i} \vee (1-k_i) z_{\alpha_i}\boldsymbol{y})\right]| \pmod 2. \end{aligned} \tag{9.115}$$

The expression

$$\Big|\bigwedge_{i=1}^{d+1}(k_i P_{\alpha_i} \vee (1-k_i) z_{\alpha_i} \boldsymbol{y})\Big| \pmod 2 \tag{9.116}$$

is equal to

$$\Big|\bigwedge_{i=1}^{d+1} z_{\alpha_i} \boldsymbol{y}\Big| \pmod 2 \tag{9.117}$$

in the case where $k_{d+1} = \ldots = k_1 = 0$, which is equal to 0 because Equation 9.112 is satisfied. Moreover, in the case where $\boldsymbol{k} \in \mathbb{Z}_2^{d+1}$ and the equations $k_{d+1} = \ldots = k_1 = 1$ and $k_{d+1} = \ldots = k_1 = 0$ are not satisfied, Expression 9.117 is equal to 0 if there exists i such that $k_i = 0$ and $z_{\alpha_i} = 0$. Otherwise it is equal to

$$\Big|\bigwedge_{i=1}^{d} P_{\beta_i} \wedge \boldsymbol{y}\Big| \pmod 2 \tag{9.118}$$

for some $\beta_1, \ldots, \beta_d$ satisfying $0 \leq \beta_1 \leq \ldots \leq \beta_d < n$. Expression 9.118 is equal to Equation 9.113 and is therefore equal to 0. Thus,

$$\Big|\bigwedge_{i=1}^{d+1}(k_i P_{\alpha_i} \vee (1-k_i) z_{\alpha_i} \boldsymbol{y})\Big| \equiv 0 \pmod 2 \tag{9.119}$$

for all $\boldsymbol{k} \in \mathbb{Z}_2^{d+1}$ such that $k_i = 0$ for some i. Finally, in the case where $k_{d+1} = \ldots = k_1 = 1$, Expression 9.116 is equal to

$$\Big|\bigwedge_{i=1}^{d+1} P_{\alpha_i}\Big| \pmod 2 \tag{9.120}$$

Therefore,

$$\begin{aligned}\Big|\bigwedge_{i=1}^{d+1} P'_{\alpha_i}\Big| &\equiv \sum_{k_{d+1}=0}^{1} \ldots \sum_{k_1=0}^{1} \Big|\left[\bigwedge_{i=1}^{d+1}(k_i P_{\alpha_i} \vee (1-k_i) z_{\alpha_i} \boldsymbol{y})\right]\Big| \pmod 2 \\ &\equiv \Big|\bigwedge_{i=1}^{d+1} P_{\alpha_i}\Big| \pmod 2.\end{aligned} \tag{9.121}$$

□

Based on Theorem 21, we can prove the following subadditivity theorem which is a generalization of Theorem 15.

Theorem 22. *Let d be a non-negative integer, let $U_1 \in \mathcal{D}_{d+1}^C$, and let $U_2 \in \mathcal{D}_{d+1}$ where $\mathcal{D}_{d+1}$ is the diagonal subgroup of the $(d+1)$th level of the Clifford hierarchy, and $\mathcal{D}_{d+1}^C$ is the subgroup of $\mathcal{D}_{d+1}$ which can be implemented using only $C^{\otimes d}Z$ gates. If $\tau[U_1], \tau[U_2] > 0$, then $\tau[U_1 \otimes U_2] < \tau[U_1] + \tau[U_2]$ where $\tau[U]$ is the optimal number of $R_Z(\pi/2^d)$ gates required to implement U without ancillary qubits over the $\{\text{CNOT}, R_Z(\pi/2^d), R_Z(2\pi/2^d)\}$ gate set.*

Proof. Let $W = \begin{bmatrix} P & Q \end{bmatrix}$ be a parity table such that P and Q are the parity tables associated with the implementation of U_1 and U_2 and which have $\tau[U_1]$ and $\tau[U_2]$ columns respectively. Then, because $U_1 \in \mathcal{D}^C_{d+1}$, P satisfies

$$\Big| \bigwedge_{i=1}^{d} P_{\alpha_i} \Big| \equiv 0 \pmod 2 \tag{9.122}$$

for all $\alpha_1, \ldots, \alpha_d$ satisfying $0 \leq \alpha_1 < \ldots \leq \alpha_d < n$ where n is the number of qubits on which U_1 is acting. Let $\boldsymbol{z} = P_{:,i} \oplus Q_{:,j}$ for any i and j satisfying $0 \leq i < \tau[U_1]$, $0 \leq j < \tau[U_2]$, and let P' be a parity table such that

$$P' = \begin{cases} P \oplus \boldsymbol{z}\mathbf{1}^T & \text{if } \tau[U_1] \equiv 0 \pmod 2, \\ \begin{bmatrix} P \oplus \boldsymbol{z}\mathbf{1}^T & \boldsymbol{z} \end{bmatrix} & \text{otherwise.} \end{cases}$$

Then, as stated by Theorem 21, the parity table P' satisfies

$$\Big| \bigwedge_{i=1}^{d+1} P'_{\alpha_i} \Big| \equiv \Big| \bigwedge_{i=1}^{d+1} P_{\alpha_i} \Big| \pmod 2 \tag{9.123}$$

for all $0 \leq \alpha_1 \leq \ldots \leq \alpha_{d+1} < n$. And so the parity table $W' = \begin{bmatrix} P' & Q \end{bmatrix}$, which has at most one more column than W, also satisfies

$$\Big| \bigwedge_{i=1}^{d+1} W'_{\alpha_i} \Big| \equiv \Big| \bigwedge_{i=1}^{d+1} W_{\alpha_i} \Big| \pmod 2 \tag{9.124}$$

for all $0 \leq \alpha_1 \leq \ldots \leq \alpha_{d+1} < n$. However, we can notice that $P'_{:,i} = Q_{:,j}$. Therefore, by removing these two columns from W' Equation 9.124 still holds and W' has at least one less column than W. The parity table W' implements the unitary $U_1 \otimes U_2$ up to an operator implementable over the $\{\text{CNOT}, R_Z(2\pi/2^d)\}$ gate set and has at most $\tau[U_1] + \tau[U_2] - 1$ columns, thus we have $\tau[U_1 \otimes U_2] \leq \tau[U_1] + \tau[U_2] - 1 < \tau[U_1] + \tau[U_2]$. □

Based on this subadditivity theorem and on Theorem 21, we can derive the following upper bound on the number of $R_Z(\pi/2^d)$ gates in a $\{\text{CNOT}, R_Z(\pi/2^d), R_Z(2\pi/2^d)\}$ circuit.

Theorem 23. *Let d be a non-negative integer. The number of $R_Z(\pi/2^d)$ gates in an n-qubit $\{\text{CNOT}, R_Z(\pi/2^d), R_Z(2\pi/2^d)\}$ circuit can be upper bounded by*

$$2 \left\lfloor \sum_{i=1}^{d} \frac{\binom{n}{i}}{2} \right\rfloor + 1 \leq \sum_{i=0}^{d} \binom{n}{i} \tag{9.125}$$

in polynomial time.

Proof. Let U be a unitary gate implementable over the $\{\text{CNOT}, R_Z(\pi/2^d), R_Z(2\pi/2^d)\}$ gate set, let P be a parity table of size $n \times m$ which implements U up to an operator implementable over the $\{\text{CNOT}, R_Z(2\pi/2^d)\}$ gate set, and let L be a matrix with rows labelled by $(\alpha_1 \ldots \alpha_k)$ such that

$$L_{\alpha_1 \ldots \alpha_k} = \bigwedge_{i=1}^{k} P_{\alpha_i} \tag{9.126}$$

for all $\alpha_1, \ldots, \alpha_k$ satisfying $0 \leq \alpha_i < \ldots < \alpha_k < n$ and k satisfying $1 \leq k \leq d$. If P has strictly more than $\sum_{i=0}^{d} \binom{n}{i}$ columns then we can necessarily find a non-zero vector $\boldsymbol{y}$ satisfying $L\boldsymbol{y} = \boldsymbol{0}$ and $\boldsymbol{y} \neq \boldsymbol{1}$ because L has $\sum_{i=1}^{d} \binom{n}{i}$ rows. Note that such vector $\boldsymbol{y}$ necessarily satisfies Equations 9.113 of Theorem 21. We can then divide P into two non-empty parity tables $P^{(1)}$ and $P^{(2)}$ where the column $P_{:,i}$ belongs to $P^{(1)}$ if and only if $y_i = 1$ and to $P^{(2)}$ otherwise. The parity tables $P^{(1)}$ and $P^{(2)}$ are implementations of some unitary gates $U_1 \in \mathcal{D}_{d+1}^{C}$ and $U_2 \in \mathcal{D}_{d+1}$ respectively. The subadditivity theorem (Theorem 22) can then be exploited to reduce the number of columns of P. Let $\boldsymbol{z} = P_{:,i} \oplus P_{:,j}$ where i and j are satisfying $y_i = 1$ and $y_j = 0$, and let

$$P' = \begin{cases} P \oplus \boldsymbol{z}\boldsymbol{y}^T & \text{if } |\boldsymbol{y}| \equiv 0 \pmod 2, \\ \begin{bmatrix} P \oplus \boldsymbol{z}\boldsymbol{y}^T & \boldsymbol{z} \end{bmatrix} & \text{otherwise.} \end{cases}$$

By Theorem 21, P' implements the same unitary gate as P up to an operator implementable over the {CNOT, $R_Z(2\pi/2^d)$} gate set. The parity table P' has at most one more column than P and we have $P'_{:,i} = P'_{:,j}$. Therefore, the columns i and j can be removed from P', which entails that P' has at least one less column than P. We showed that if the number of columns of P is strictly greater than $\sum_{i=0}^{d} \binom{n}{i}$, then the number of columns of P can be reduced by at least one in polynomial time. Moreover, if the number of columns of P is equal to $\sum_{i=0}^{d} \binom{n}{i}$ and is even, then we can necessarily find a non-zero vector $\boldsymbol{y}$ satisfying $L\boldsymbol{y} = \boldsymbol{0}$ because L has $\sum_{i=1}^{d} \binom{n}{i}$ rows. If there exist i such that $y_i = 0$ then we can reduced the number of columns of P as described above. Otherwise we must have $|\boldsymbol{y}| \equiv 0 \pmod 2$, and so $\boldsymbol{y}$ satisfies the Equations of Theorem 21. Therefore, if $P' = P \oplus \boldsymbol{z}\boldsymbol{y}^T$ where $\boldsymbol{z}$ is equal to the ith column of P for any i, then P' implements the same unitary gate as P up to an operator implementable over the {CNOT, $R_Z(2\pi/2^d)$} gate set. The parity table P' has the same number of columns as P but its ith column is equal to the null vector and can therefore be removed, which leads to a parity table containing $\sum_{i=1}^{d} \binom{n}{i}$ columns.

The polynomial-time procedure described above to reduce the number of columns of P can be repeated until P has a number of columns lower or equal to

$$2 \left\lfloor \sum_{i=1}^{d} \frac{\binom{n}{i}}{2} \right\rfloor + 1 \leq \sum_{i=0}^{d} \binom{n}{i}. \tag{9.127}$$

□

The overall complexity of the algorithm described in the proof of Theorem 23 is $\mathcal{O}(n^{2d+1}m)$ where m is the number of $R_Z(\pi/2^d)$ gates in the initial circuit. For $d = 0$, the algorithm yields a parity table which contains at most one column, which is optimal. For $d = 1$, the algorithm corresponds to Lempel's matrix factorization algorithm [167], which is optimal. For $d = 2$, the algorithm is similar to the `TOHPE` algorithm (Algorithm 13). Note that the `TOHPE` algorithm has a complexity of $\mathcal{O}(n^2m^3)$ instead of $\mathcal{O}(n^5m)$ because of the heuristic used by the algorithm to better optimize the number of T gates. The same heuristic can be used in the case where $d > 2$, which would lead to an algorithm having a complexity of $\mathcal{O}(n^d m^3)$.

9.4.3 Generalization of the `FastTODD` algorithm for optimizing $R_Z(\pi/2^d)$ gates

In this subsection we generalize the `FastTODD` algorithm (Algorithm 14) for the optimization of the number of $R_Z(\pi/2^d)$ gates in a {CNOT, $R_Z(\pi/2^d)$, $R_Z(2\pi/2^d)$} circuit.

Theorem 24. *Let d be a non-negative integer, let P be a parity table of size $n \times m$ and $P' = P \oplus \boldsymbol{z}\boldsymbol{y}^T$ where $\boldsymbol{z}$ and $\boldsymbol{y}$ are vectors of size n and m respectively and such that $|\boldsymbol{y}| \equiv 0 \pmod 2$. And let L and X be matrices and $\boldsymbol{v}$ be a vector, all with rows labelled by $(\alpha_1 \dots \alpha_k)$ such that*

$$L_{\alpha_1 \dots \alpha_k} = \bigwedge_{i=1}^{k} P_i \tag{9.128}$$

$$X_{\alpha_1 \dots \alpha_k, \beta_1 \dots \beta_{k-1}} = \bigvee_{i=1}^{k} z_{\alpha_i} \delta_{\alpha_1 \dots \alpha_{i-1} \alpha_{i+1} \dots \alpha_k, \beta_1 \dots \beta_{k-1}} \tag{9.129}$$

$$v_{\alpha_1 \dots \alpha_k} = \bigwedge_{i=1}^{k} z_{\alpha_i} \tag{9.130}$$

for all $\alpha_1, \dots, \alpha_k$ and $\beta_1, \dots, \beta_{k-1}$ satisfying $0 \leq \alpha_1 < \dots < \alpha_k < n$ and $0 \leq \beta_1 < \dots < \beta_{k-1} < n$ where k satisfies $1 \leq k \leq d$, and where δ is defined as follows:

$$\delta_{\alpha_1 \dots \alpha_{i-1} \alpha_{i+1} \dots \alpha_k, \beta_1 \dots \beta_{k-1}} = \begin{cases} 1 & \text{if } \alpha_1 = \beta_1, \dots, \alpha_{i-1} = \beta_{i-1}, \alpha_{i+1} = \beta_i, \dots, \alpha_k = \beta_{k-1}, \\ 0 & \text{otherwise.} \end{cases} \tag{9.131}$$

All entries of L, X and $\boldsymbol{v}$ are set to zero in the case where $d = 0$ for L and in the case where $d \leq 1$ for X and $\boldsymbol{v}$. Then, the equality

$$\Big|\bigwedge_{i=1}^{k} P'_{\alpha_i}\Big| \equiv \Big|\bigwedge_{i=1}^{k} P_{\alpha_i}\Big| \pmod 2 \tag{9.132}$$

holds for all $0 \leq \alpha_1 < \dots < \alpha_k < n$ and $1 \leq k \leq d+1$ if and only if there exists a vector $\boldsymbol{y}'$ and a Boolean b such that $L\boldsymbol{y} \oplus X\boldsymbol{y}' \oplus b\boldsymbol{v} = \boldsymbol{0}$.

Proof. In the case where $k = 1$, we have

$$\begin{aligned} |P'_{\alpha_1}| &= |P_{\alpha_1} \oplus z_{\alpha_1}\boldsymbol{y}| \\ &\equiv |P_{\alpha_1}| + |z_{\alpha_1}\boldsymbol{y}| \pmod 2 \\ &\equiv |P_{\alpha_1}| \pmod 2 \end{aligned} \tag{9.133}$$

because $|\boldsymbol{y}| \equiv 0 \pmod 2$. This proves Theorem 24 in the case where $d = 0$, because in such case the equation $L\boldsymbol{y} \oplus X\boldsymbol{y}' \oplus b\boldsymbol{v} = \boldsymbol{0}$ is always satisfied. We now prove Theorem 24 in the case where $d > 0$ and $\boldsymbol{z}$ is satisfying $|\boldsymbol{z}| = 1$. Without loss of generality we will assume that $z_{\alpha_1} = 1$ and $z_{\alpha_i} = 0$ for all i such that $i \neq 1$, then

$$\begin{aligned} \Big|\bigwedge_{i=1}^{k} P'_{\alpha_i}\Big| &= \Big|(P_{\alpha_1} \oplus z_{\alpha_1}\boldsymbol{y}) \bigwedge_{i=2}^{k} P_{\alpha_i}\Big| \\ &= \Big|(P_{\alpha_1} \oplus \boldsymbol{y}) \bigwedge_{i=2}^{k} P_{\alpha_i}\Big| \\ &= \Big|\bigwedge_{i=1}^{k} P_{\alpha_i} \oplus \bigwedge_{i=2}^{k} P_{\alpha_i} \wedge \boldsymbol{y}\Big| \\ &\equiv \Big|\bigwedge_{i=1}^{k} P_{\alpha_i}\Big| + \Big|\bigwedge_{i=2}^{k} P_{\alpha_i} \wedge \boldsymbol{y}\Big| \pmod 2 \end{aligned} \tag{9.134}$$

where $\alpha_1, \ldots, \alpha_k$ are satisfying $0 \leq \alpha_1 < \ldots < \alpha_k < n$ and k is satisfying $1 \leq k \leq d+1$. Therefore, proving Theorem 24 in the case where $d > 0$ and $|\boldsymbol{z}| = 1$ can be done by showing that there exists a vector $\boldsymbol{y'}$ and a Boolean b such that $L\boldsymbol{y} \oplus X\boldsymbol{y'} \oplus b\boldsymbol{v} = \boldsymbol{0}$ if and only if the equation

$$\Big|\bigwedge_{i=2}^{k} P_{\alpha_i} \wedge \boldsymbol{y}\Big| \equiv 0 \pmod 2 \tag{9.135}$$

holds for all $\alpha_2, \ldots, \alpha_k$ satisfying $\alpha_1 < \alpha_2 < \ldots < \alpha_k < n$ and k satisfying $2 \leq k \leq d+1$. By definition we have $X_{\alpha_2 \ldots \alpha_k} = \boldsymbol{0}$ and $v_{\alpha_2 \ldots \alpha_k} = 0$ for all $\alpha_2, \ldots, \alpha_k$ satisfying $0 \leq \alpha_1 < \ldots < \alpha_k < n$ and k satisfying $2 \leq k \leq d+1$. Hence,

$$L_{\alpha_2 \ldots \alpha_k}\boldsymbol{y} \oplus X_{\alpha_2 \ldots \alpha_k}\boldsymbol{y'} \oplus bv_{\alpha_2 \ldots \alpha_k} = L_{\alpha_2 \ldots \alpha_k}\boldsymbol{y} = \Big|\bigwedge_{i=2}^{k} P_{\alpha_i} \wedge \boldsymbol{y}\Big| \pmod 2 \tag{9.136}$$

and so if $L\boldsymbol{y} \oplus X\boldsymbol{y'} \oplus b\boldsymbol{v} = \boldsymbol{0}$ then $L_{\alpha_2 \ldots \alpha_k}\boldsymbol{y} = 0$ which imply that Equation 9.135 is satisfied. Conversely, if Equation 9.135 is satisfied then we must have

$$L_{\alpha_2 \ldots \alpha_k}\boldsymbol{y} \oplus X_{\alpha_2 \ldots \alpha_k}\boldsymbol{y'} \oplus b\boldsymbol{v}_{\alpha_2 \ldots \alpha_k} = L_{\alpha_2 \ldots \alpha_k} = \boldsymbol{0} \tag{9.137}$$

for all $\alpha_2, \ldots, \alpha_k$ satisfying $\alpha_1 < \alpha_2 < \ldots < \alpha_k < n$ and k satisfying $2 \leq k \leq d+1$. Moreover we have

$$L_{\alpha_1}\boldsymbol{y} \oplus X_{\alpha_1}\boldsymbol{y'} \oplus bv_{\alpha_1} = L_{\alpha_1}\boldsymbol{y} \oplus b \tag{9.138}$$

because $X_{\alpha_1} = \boldsymbol{0}$ and $v_{\alpha_1} = 1$. And

$$L_{\alpha_1 \ldots \alpha_k}\boldsymbol{y} \oplus X_{\alpha_1 \ldots \alpha_k}\boldsymbol{y'} \oplus bv_{\alpha_1 \ldots \alpha_k} = L_{\alpha_1 \ldots \alpha_k}\boldsymbol{y} \oplus y'_{\alpha_2 \ldots \alpha_k} \tag{9.139}$$

for all $\alpha_2, \ldots, \alpha_k$ satisfying $\alpha_1 < \alpha_2 < \ldots < \alpha_k < n$ and k satisfying $2 \leq k \leq d$, because $v_{\alpha_1 \ldots \alpha_k} = 0$ and

$$X_{\alpha_1 \ldots \alpha_k}\boldsymbol{y'} = \bigoplus_{i=1}^{k} z_{\alpha_i} y'_{\alpha_1 \ldots \alpha_{i-1}\alpha_{i+1} \ldots \alpha_k} \tag{9.140}$$

for all $\alpha_1, \ldots, \alpha_k$ satisfying $0 \leq \alpha_1 < \ldots < \alpha_k < n$ and k satisfying $2 \leq k \leq d$ and where the rows of $\boldsymbol{y'}$ are labelled the same way as the columns of X. Let $\boldsymbol{y'}$ be such that $y'_{\alpha_2 \ldots \alpha_k} = L_{\alpha_1 \ldots \alpha_k}\boldsymbol{y}$ for all $\alpha_2, \ldots, \alpha_k$ satisfying $\alpha_1 < \alpha_2 < \ldots < \alpha_k < n$ and k satisfying $2 \leq k \leq d$ and b be such that $b = L_{\alpha_1}\boldsymbol{y}$, then we have $L\boldsymbol{y} \oplus X\boldsymbol{y'} \oplus b\boldsymbol{v} = \boldsymbol{0}$. Thus, we proved that

$$L\boldsymbol{y} \oplus X\boldsymbol{y'} \oplus b\boldsymbol{v} = \boldsymbol{0} \iff \Big|\bigwedge_{i=2}^{k} P_{\alpha_i} \wedge \boldsymbol{y}\Big| \equiv 0 \pmod 2 \tag{9.141}$$

for all $\alpha_2, \ldots, \alpha_k$ satisfying $\alpha_1 < \alpha_2 < \ldots < \alpha_k < n$ and k satisfying $2 \leq k \leq d+1$, and so Theorem 24 is true in the case where $|\boldsymbol{z}| = 1$.

Let B be a full rank binary matrix of size $n \times n$ which represents a change of basis, and let $\tilde{L}, \tilde{X}$ and $\tilde{\boldsymbol{v}}$ be constructed in the same way as L, X and $\boldsymbol{v}$ but with respect to BP and $B\boldsymbol{z}$. Notice that if Equation 9.132 is satisfied for P and $\boldsymbol{z}$ then it is also satisfied for BP and $B\boldsymbol{z}$. The proof of Theorem 24 can then be completed by proving that if there exists a vector $\boldsymbol{y'}$ and a Boolean b such that $L\boldsymbol{y} \oplus X\boldsymbol{y'} \oplus b\boldsymbol{v} = \boldsymbol{0}$ then there exists a vector $\tilde{\boldsymbol{y}}'$ and a Boolean $\tilde{b}$ such that $\tilde{L}\boldsymbol{y} \oplus \tilde{X}\tilde{\boldsymbol{y}}' \oplus \tilde{b}\tilde{\boldsymbol{v}} = \boldsymbol{0}$. Suppose this proposition to be true and let B be such that $|B\boldsymbol{z}| = 1$. Then, as previously demonstrated, Equation 9.132 holds for BP and $B\boldsymbol{z}$ if and only if there exists a

vector $\tilde{\boldsymbol{y}}'$ and a Boolean $\tilde{b}$ such that $\tilde{L}\boldsymbol{y} \oplus \tilde{X}\tilde{\boldsymbol{y}}' \oplus \tilde{b}\tilde{\boldsymbol{v}} = \boldsymbol{0}$, which would imply that there exists a vector $\boldsymbol{y}'$ and a Boolean b such that $L\boldsymbol{y} \oplus X\boldsymbol{y}' \oplus b\boldsymbol{v} = \boldsymbol{0}$ if and only if Equation 9.132 holds for P and $\boldsymbol{z}$.

Let $\tilde{\boldsymbol{z}}$ and $\tilde{P}$ be such that $\tilde{z}_{\alpha_1} = z_{\alpha_1} \oplus z_{\alpha_2}$ and $\tilde{P}_{\alpha_1} = P_{\alpha_1} \oplus P_{\alpha_2}$ for some fixed α_1, α_2 satisfying $\alpha_1 \neq \alpha_2$ and $\tilde{z}_\beta = z_\beta$, $\tilde{P}_\beta = P_\beta$ for all β such that $\beta \neq \alpha_1$. Without loss of generality, we will assume that $\alpha_1 = 0$ and $\alpha_2 = 1$. Indeed, all the other possibilities can be reduced to this case simply by permuting the rows of the parity table P and the vector $\boldsymbol{z}$. Let $\tilde{L}$, $\tilde{X}$ and $\tilde{\boldsymbol{v}}$ be constructed in the same way as L, X and $\boldsymbol{v}$ but with respect to $\tilde{P}$ and $\tilde{\boldsymbol{z}}$. We will now prove that if there exists a vector $\boldsymbol{y}'$ and a Boolean b such that $L\boldsymbol{y} \oplus X\boldsymbol{y}' \oplus b\boldsymbol{v} = \boldsymbol{0}$ then $\tilde{L}\boldsymbol{y} \oplus \tilde{X}\tilde{\boldsymbol{y}}' \oplus b\tilde{\boldsymbol{v}} = \boldsymbol{0}$, where $\tilde{\boldsymbol{y}}'$ satisfies $\tilde{y}'_{\alpha_1\alpha_3\ldots\alpha_k} = y'_{\alpha_1\alpha_3\ldots\alpha_k} \oplus y'_{\alpha_2\ldots\alpha_k}$ and $\tilde{y}'_{\alpha_3\ldots\alpha_k} = y'_{\alpha_3\ldots\alpha_k}$ such that $\alpha_i \neq \alpha_1$ for all i. By using Equation 9.140 we can deduce that

$$\tilde{X}_{\alpha_3\ldots\alpha_k}\tilde{\boldsymbol{y}}' = \bigoplus_{i=3}^{k} \tilde{z}_{\alpha_i}\tilde{y}'_{\alpha_3\ldots\alpha_{i-1}\alpha_{i+1}\ldots\alpha_k} = \bigoplus_{i=3}^{k} z_{\alpha_i} y'_{\alpha_3\ldots\alpha_{i-1}\alpha_{i+1}\ldots\alpha_k} = X_{\alpha_3\ldots\alpha_k}\boldsymbol{y}' \tag{9.142}$$

for all $\alpha_3, \ldots, \alpha_k$ satisfying $\alpha_2 < \alpha_3 < \ldots < \alpha_k < n$ and k satisfying $3 \leq k \leq d+2$, and where the rows of $\tilde{y}'$ and y' are labelled in the same as the columns of $\tilde{X}$ and X. And from the definitions of the vectors $\tilde{\boldsymbol{v}}$ and $\boldsymbol{v}$, we can deduce that

$$\tilde{v}_{\alpha_3\ldots\alpha_k} = \bigwedge_{i=3}^{k} \tilde{z}_{\alpha_i} = \bigwedge_{i=3}^{k} z_{\alpha_i} = v_{\alpha_3\ldots\alpha_k} \tag{9.143}$$

for all $\alpha_3, \ldots, \alpha_k$ satisfying $\alpha_2 < \alpha_3 < \ldots < \alpha_k < n$ and k satisfying $3 \leq k \leq d+2$. Equations 9.142 and 9.143 imply that

$$\tilde{L}_{\alpha_3\ldots\alpha_k}\boldsymbol{y} \oplus \tilde{X}_{\alpha_3\ldots\alpha_k}\tilde{\boldsymbol{y}}' \oplus b\tilde{v}_{\alpha_3\ldots\alpha_k} = L_{\alpha_3\ldots\alpha_k}\boldsymbol{y} \oplus X_{\alpha_3\ldots\alpha_k}\boldsymbol{y}' \oplus bv_{\alpha_3\ldots\alpha_k} = 0 \tag{9.144}$$

for all $\alpha_3, \ldots, \alpha_k$ satisfying $\alpha_2 < \alpha_3 < \ldots < \alpha_k < n$ and k satisfying $3 \leq k \leq d+2$. Furthermore, we have

$$\begin{aligned}
\tilde{L}_{\alpha_1\ldots\alpha_k}\boldsymbol{y} &\equiv \Big|\bigwedge_{i=1}^{k} \tilde{P}_{\alpha_i} \wedge \boldsymbol{y}\Big| \pmod 2 \\
&\equiv \Big|(P_{\alpha_1} \oplus P_{\alpha_2}) \wedge \bigwedge_{i=2}^{k} P_{\alpha_k} \wedge \boldsymbol{y}\Big| \pmod 2 \\
&\equiv \Big|\bigwedge_{i=1}^{k} P_{\alpha_k} \wedge \boldsymbol{y}\Big| + \Big|\bigwedge_{i=2}^{k} P_{\alpha_k} \wedge \boldsymbol{y}\Big| \pmod 2 \\
&\equiv L_{\alpha_1\ldots\alpha_k}\boldsymbol{y} + L_{\alpha_2\ldots\alpha_k}\boldsymbol{y} \pmod 2 \\
&\equiv X_{\alpha_1\ldots\alpha_k}\boldsymbol{y}' + bv_{\alpha_1\ldots\alpha_k} + X_{\alpha_2\ldots\alpha_k}\boldsymbol{y}' + bv_{\alpha_2\ldots\alpha_k} \pmod 2
\end{aligned} \tag{9.145}$$

for all $\alpha_3, \ldots, \alpha_k$ satisfying $\alpha_2 < \alpha_3 < \ldots < \alpha_k < n$ and k satisfying $3 \leq k \leq d$, which entails

$$\begin{aligned}
&\tilde{L}_{\alpha_1\ldots\alpha_k}\boldsymbol{y} \oplus \tilde{X}_{\alpha_1\ldots\alpha_k}\tilde{\boldsymbol{y}}' \oplus b\tilde{v}_{\alpha_1\ldots\alpha_k} \\
&= X_{\alpha_1\ldots\alpha_k}\boldsymbol{y}' \oplus X_{\alpha_2\ldots\alpha_k}\boldsymbol{y}' \oplus \tilde{X}_{\alpha_1\ldots\alpha_k}\tilde{\boldsymbol{y}}' \oplus bv_{\alpha_1\ldots\alpha_k} \oplus bv_{\alpha_2\ldots\alpha_k} \oplus b\tilde{v}_{\alpha_1\ldots\alpha_k} \\
&= \bigoplus_{i=1}^{k} z_{\alpha_i} y'_{\alpha_1\ldots\alpha_{i-1}\alpha_{i+1}\ldots\alpha_k} \oplus \bigoplus_{i=2}^{k} z_{\alpha_i} y'_{\alpha_2\ldots\alpha_{i-1}\alpha_{i+1}\ldots\alpha_k} \oplus \bigoplus_{i=1}^{k} \tilde{z}_{\alpha_i}\tilde{y}'_{\alpha_1\ldots\alpha_{i-1}\alpha_{i+1}\ldots\alpha_k} \\
&\quad \oplus b\bigwedge_{i=1}^{k} z_{\alpha_i} \oplus b\bigwedge_{i=2}^{k} z_{\alpha_i} \oplus b\bigwedge_{i=1}^{k} \tilde{z}_{\alpha_i}
\end{aligned}$$

$$
\begin{aligned}
&= \bigoplus_{i=1}^{k} z_{\alpha_i} y'_{\alpha_1 \ldots \alpha_{i-1}\alpha_{i+1}\ldots\alpha_k} \oplus \bigoplus_{i=2}^{k} z_{\alpha_i} y'_{\alpha_2 \ldots \alpha_{i-1}\alpha_{i+1}\ldots\alpha_k} \oplus (z_{\alpha_1} \oplus z_{\alpha_2}) y'_{\alpha_2\ldots\alpha_k} \\
&\quad \oplus \bigoplus_{i=2}^{k} z_{\alpha_i} \tilde{y}'_{\alpha_1 \ldots \alpha_{i-1}\alpha_{i+1}\ldots\alpha_k} \oplus b \bigwedge_{i=1}^{k} z_{\alpha_i} \oplus b \bigwedge_{i=2}^{k} z_{\alpha_i} \oplus b \bigwedge_{i=2}^{k} z_{\alpha_i} \wedge (z_{\alpha_1} \oplus z_{\alpha_2}) \\
&= \bigoplus_{i=2}^{k} z_{\alpha_i} (y'_{\alpha_1 \ldots \alpha_{i-1}\alpha_{i+1}\ldots\alpha_k} \oplus y'_{\alpha_2 \ldots \alpha_{i-1}\alpha_{i+1}\ldots\alpha_k}) \oplus z_{\alpha_2} y'_{\alpha_2\ldots\alpha_k} \\
&\quad \oplus \bigoplus_{i=2}^{k} z_{\alpha_i} \tilde{y}'_{\alpha_1 \ldots \alpha_{i-1}\alpha_{i+1}\ldots\alpha_k} \oplus b \bigwedge_{i=1}^{k} z_{\alpha_i} \oplus b \bigwedge_{i=2}^{k} z_{\alpha_i} \oplus b \bigwedge_{i=1}^{k} z_{\alpha_i} \oplus b \bigwedge_{i=2}^{k} z_{\alpha_i} \\
&= \bigoplus_{i=2}^{k} z_{\alpha_i} (y'_{\alpha_1 \ldots \alpha_{i-1}\alpha_{i+1}\ldots\alpha_k} \oplus y'_{\alpha_2 \ldots \alpha_{i-1}\alpha_{i+1}\ldots\alpha_k}) \oplus z_{\alpha_2} y'_{\alpha_2\ldots\alpha_k} \oplus z_{\alpha_2} \tilde{y}'_{\alpha_1\alpha_3\ldots\alpha_k} \\
&\quad \oplus \bigoplus_{i=3}^{k} z_{\alpha_i} \tilde{y}'_{\alpha_1 \ldots \alpha_{i-1}\alpha_{i+1}\ldots\alpha_k} \\
&= \bigoplus_{i=2}^{k} z_{\alpha_i} (y'_{\alpha_1 \ldots \alpha_{i-1}\alpha_{i+1}\ldots\alpha_k} \oplus y'_{\alpha_2 \ldots \alpha_{i-1}\alpha_{i+1}\ldots\alpha_k}) \oplus z_{\alpha_2} y'_{\alpha_2\ldots\alpha_k} \oplus z_{\alpha_2} (y'_{\alpha_1\alpha_3\ldots\alpha_k} \oplus y'_{\alpha_2\ldots\alpha_k} \\
&\quad \oplus y'_{\alpha_3\ldots\alpha_k}) \oplus \bigoplus_{i=3}^{k} z_{\alpha_i} (y'_{\alpha_1 \ldots \alpha_{i-1}\alpha_{i+1}\ldots\alpha_k} \oplus y'_{\alpha_2 \ldots \alpha_{i-1}\alpha_{i+1}\ldots\alpha_k}) \\
&= 0.
\end{aligned} \tag{9.146}
$$

Finally, we have

$$
\begin{aligned}
\tilde{L}_{\alpha_1}\boldsymbol{y} \oplus \tilde{X}_{\alpha_1}\tilde{\boldsymbol{y}}' \oplus b\tilde{v}_{\alpha_1} &= \tilde{L}_{\alpha_1}\boldsymbol{y} \oplus b\tilde{z}_{\alpha_1} \\
&= \tilde{L}_{\alpha_1}\boldsymbol{y} \oplus b(z_{\alpha_1} \oplus z_{\alpha_2}) \\
&\equiv |\tilde{P}_{\alpha_1} \wedge \boldsymbol{y}| + bv_{\alpha_1} + bv_{\alpha_2} \pmod 2 \\
&\equiv |(P_{\alpha_1} \oplus P_{\alpha_2}) \wedge \boldsymbol{y}| + bv_{\alpha_1} + bv_{\alpha_2} \pmod 2 \\
&\equiv |P_{\alpha_1} \wedge \boldsymbol{y}| + |P_{\alpha_2} \wedge \boldsymbol{y}| + bv_{\alpha_1} + bv_{\alpha_2} \pmod 2 \\
&= L_{\alpha_1} \oplus bv_{\alpha_1} \oplus L_{\alpha_2} \oplus bv_{\alpha_2} \\
&= 0.
\end{aligned} \tag{9.147}
$$

Thus, we proved that

$$
L\boldsymbol{y} \oplus X\boldsymbol{y}' \oplus b\boldsymbol{v} = \boldsymbol{0} \implies \tilde{L}\boldsymbol{y} \oplus \tilde{X}\tilde{\boldsymbol{y}}' \oplus b\tilde{v} = \boldsymbol{0} \tag{9.148}
$$

which concludes the proof of Theorem 24. □

9.5 $R_Z(\pi/2^d)$-count upper bound in universal gate sets

In this section we demonstrate an upper bound achievable in polynomial time for the $R_Z(\pi/2^d)$-count within a {CNOT, H, $R_Z(\pi/2^d)$, $R_Z(2\pi/2^d)$} circuit where d is a non-negative integer. Note that it corresponds to an upper bound for the number of T gates in a Clifford+T circuit in the case where $d = 2$. Our proof for the upper bound will partially rest on the following lemma, which has already been proven with a different approach in Reference [157] for the case where $d = 2$.

Lemma 5. *Let d be a non-negative integer and let $U \in \mathcal{D}_{d+1}$ act on n qubits where $\mathcal{D}_{d+1}$ is the diagonal subgroup of the $(d+1)$th level of the Clifford hierarchy. Then U can be decomposed into two unitary gates $U = \tilde{U}\tilde{U}'$ which can be found in polynomial time and such that $\tilde{U} \in \mathcal{D}_{d+1}$ is acting on $n-1$ qubits. Furthermore, we can find a $\{\text{CNOT}, R_Z(\pi/2^d), R_Z(2\pi/2^d)\}$ circuit implementing $\tilde{U}' \in \mathcal{D}_{d+1}$ with no more than*

$$\sum_{i=0}^{d-1} \binom{n-1}{i} + 1 \tag{9.149}$$

$R_Z(\pi/2^d)$ gates in polynomial time.

Proof. Let P be a parity table associated with an implementation of U, let β be the qubit on which $\tilde{U}$ is not acting, and let $\tilde{P}'$ be a parity table such that

$$|\bigwedge_{i=1}^{d} \tilde{P}'_{\alpha_i}| \equiv |\bigwedge_{i=1}^{d} P_{\alpha_i} \wedge P_\beta| \pmod 2 \tag{9.150}$$

for all $\alpha_1, \ldots, \alpha_d$ satisfying $0 \leq \alpha_1 \leq \ldots \leq \alpha_d < n$ and $\alpha_i \neq \beta$ for all i. The row β of $\tilde{P}'$ can be ignored, and, as stated by Theorem 23, we can optimize the number of columns of $\tilde{P}'$ such that it is lower or equal to

$$\sum_{i=0}^{d-1} \binom{n-1}{i} \tag{9.151}$$

and still satisfies Equation 9.150. If we add a null column to $\tilde{P}'$ in the case where the number of columns of $\tilde{P}'$ is not equal to $|P_\beta| \pmod 2$, and then set $\tilde{P}'_\beta = \mathbf{1}$, then we have

$$|\bigwedge_{i=1}^{d} \tilde{P}'_{\alpha_i} \wedge \tilde{P}'_\beta| = |\bigwedge_{i=1}^{d} \tilde{P}'_{\alpha_i}| \equiv |\bigwedge_{i=1}^{d} P_{\alpha_i} \wedge P_\beta| \pmod 2 \tag{9.152}$$

for all $0 \leq \alpha_1 \leq \ldots \leq \alpha_d < n$ and the number of columns of $\tilde{P}$ is at most

$$\sum_{i=0}^{d-1} \binom{n-1}{i} + 1. \tag{9.153}$$

Finally, we can easily find a parity table $\tilde{P}$ such that $\tilde{P}_\beta = \mathbf{0}$ and

$$|\bigwedge_{i=1}^{d+1} \tilde{P}_{\alpha_i}| \equiv |\bigwedge_{i=1}^{d+1} \tilde{P}'_{\alpha_i}| + |\bigwedge_{i=1}^{d+1} P_{\alpha_i}| \pmod 2 \tag{9.154}$$

for all $0 \leq \alpha_1 \leq \ldots \leq \alpha_d < n$, which implies its associated unitary gate $\tilde{U}$ doesn't act on qubit β. Thus, $\tilde{U}$ and $\tilde{U}'$ are forming a decomposition of U up to an operator V implementable over $\{\text{CNOT}, R_Z(2\pi/2^d)\}$ gate set: $U = \tilde{U}\tilde{U}'V$, where $\tilde{U}'$ is the unitary gate associated with the parity table $\tilde{P}'$. The operator V can then be merged with the unitary $\tilde{U}'$ to comply with the decomposition of U established in Lemma 5. □

Based on Lemma 5 and Theorem 23, we can prove the following theorem which provides an upper bound for the number of $R_Z(\pi/2^d)$ gates in a $\{\text{CNOT}, H, R_Z(\pi/2^d), R_Z(2\pi/2^d)\}$ circuit. Note that in the case where $d = 2$, we get an upper bound of $(n+1)(n+2h)/2+1$ for the number of T gates in a Clifford+T circuit, which is significantly better than the previously best-known ancilla-free upper bound of $\mathcal{O}(n^2h)$ [106], where n is the number of qubits and h is the number of Hadamard gates in the circuit.

Theorem 25. *Let U be a unitary gate acting on n qubits and implementable over the* {CNOT, H, $R_Z(\pi/2^d)$, $R_Z(2\pi/2^d)$} *gate set where d is a non-negative integer and with h internal Hadamard gates such that*

$$U = C_f U_0 \prod_{i=1}^{h} \left[H_{\alpha_i} U_i \right] C_{init} \tag{9.155}$$

where C_f, C_{init} are implementable over the {CNOT, H, $R_Z(2\pi/2^d)$} *gate set, H_{α_i} denotes the Hadamard gate applied on some qubit α_i, and $U_0, \ldots, U_h$ are implementable over the* {CNOT, $R_Z(\pi/2^d)$, $R_Z(2\pi/2^d)$} *gate set. Then we can find a implementation of U over the* {CNOT, H, $R_Z(\pi/2^d)$, $R_Z(2\pi/2^d)$} *gate set, in polynomial time, and such that the number of $R_Z(\pi/2^d)$ gates is lower or equal to*

$$\sum_{i=0}^{d} \binom{n}{i} + h \left(\sum_{i=0}^{d-1} \binom{n-1}{i} + 1 \right). \tag{9.156}$$

Proof. As stated by Lemma 5, the unitary U_h can be decomposed into two unitary gates: $U_h = \tilde{U}_h \tilde{U}'_h$, such that $\tilde{U}_h$ is not acting on qubit α_h, which imply that $\tilde{U}_h$ commutes with the hth internal Hadamard gate. Let $U_i \tilde{U}_{i+1} = \tilde{U}_i \tilde{U}'_i$ for all i satisfying $0 < i < h$ where $\tilde{U}_i \tilde{U}'_i$ is a decomposition of $U_i \tilde{U}'_{i+1}$ such as given by Lemma 5 where $\tilde{U}'_i$ is not acting on qubit α_i. Then Equation 9.155 is equivalent to

$$U = C_f U_0 \tilde{U}_1 \prod_{i=1}^{h} \left[H_{\alpha_i} \tilde{U}'_i \right] C_{init} \tag{9.157}$$

and, as stated by Lemma 5, we can find an implementation of $\tilde{U}'_i$ in polynomial time and with no more than

$$\sum_{k=0}^{d-1} \binom{n-1}{k} + 1 \tag{9.158}$$

$R_Z(\pi/2^d)$ gates, for all i satisfying $0 < i \leq h$. Moreover, as stated by Theorem 23, we can find an implementation of $U_0 \tilde{U}_1$ in polynomial time and with no more than

$$\sum_{i=0}^{d} \binom{n}{i} \tag{9.159}$$

$R_Z(\pi/2^d)$ gates. Thus, U can be implemented in polynomial time over the {CNOT, H, $R_Z(\pi/2^d)$, $R_Z(2\pi/2^d)$} gate set and with a number of $R_Z(\pi/2^d)$ gates that is lower or equal to

$$\sum_{i=0}^{d} \binom{n}{i} + h \left(\sum_{i=0}^{d-1} \binom{n-1}{i} + 1 \right). \tag{9.160}$$

□

Part IV

Optimization of Fault-Tolerant Quantum Computing using ZX-Calculus and Graph Theory

Chapter 10

Hadamard Gate Degadgetization

In this chapter, we present a method for optimizing the number of qubits in a quantum circuit while preserving the number of non-Clifford gates. Our approach is based on the degadgetization of Hadamard gates, which is the reverse process of Hadamard gate gadgetization as depicted in Figure 3.2. This approach is particularly useful for Clifford+T circuits in which the number of T gates has been optimized. Indeed, the T-count optimizers yielding the best performances are relying on this measurement-based gadget to circumvent Hadamard gates and better optimize the number of T gates [108, 112, 131], such as done in Chapter 9. In some cases, this can drastically increase the number of qubits required to implement the optimized quantum circuit. This can be impractical simply because the number of ancillary qubits at disposal may be limited, or it can counteract the gains achieved through T-count optimization. For instance, in the surface code, implementing the T gate fault-tolerantly can be done via magic state distillation, which is a procedure inducing an overhead in time and in the number of qubits [38]. However, it has been shown that for some compilers designed for the surface code, the overhead of magic state distillation can be significantly lower than the overhead induced by the surface code for encoding reliable logical qubits during the whole computation, especially for large quantum circuits [168]. In this context, increasing the number of qubits to lower the number of T gates can be less impactful, or, in some cases, counterproductive.

An approach for lowering the number of qubits required by this Hadamard gate gadgetization procedure consists in optimizing the number of Hadamard gates in the initial circuit, as done in Chapter 6 or Reference [112]. The optimization process then consists in optimizing the number of Hadamard gates, gadgetizing them, and optimizing the number of T gates. To further lower the overhead caused by this Hadamard gate gadgetization strategy, we propose to append a final step to this process, called Hadamard gate degadgetization, which consists in reversing, as much as possible, the gadgetization of Hadamard gates. We will show that if certain conditions are satisfied, then it is possible to undo the gadgetization of a Hadamard gate and reduce the number of auxiliary qubits by one without increasing the T-count. To the best of our knowledge, this method for optimizing the number of qubits has not been previously considered.

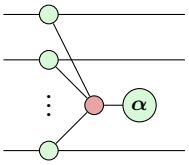
In Section 10.1, we introduce the Hadamard gate degadgetization problem. Then, in Section 10.2, we present an algorithm for degadgetizing a maximal number of Hadamard gates in a given quantum circuit. In particular, we show how the Hadamard gate degadgetization problem relates to the feedback vertex set problem, a well-known problem in graph theory. Finally, we prove the NP-hardness of the problem in Section 10.3. Benchmarks are provided in Section 11.3 of the next chapter.

10.1 Problem statement

In this section, we formulate the problem of optimizing the number of qubits in a quantum circuit through the degadgetization of Hadamard gates. We will consider circuits in which all Hadamard gates have been gadgetized. For example, that is the case of the circuits in which the number of T gates has been optimized by the algorithms of References [108, 112], or by the algorithm presented in Chapter 9. The operations composing this kind of circuit can be represented by diagonal Pauli rotations, classically controlled Clifford gates and a final Clifford operator. A diagonal Pauli rotation $R_P(\alpha)$ acting on n qubits is defined as

$$R_P(\alpha) = \exp(-i\alpha P/2) \tag{10.1}$$

where $P \in \{I, Z\}^{\otimes n}$ is a Pauli product, I is the identity matrix, and Z is the Pauli Z matrix. We will use P_i to denote the ith Pauli matrix of a Pauli product P. A diagonal Pauli rotation can be represented by a phase gadget in ZX-calculus:



(10.2)

We can notice that two phase gadgets necessarily commute. Thus, if a circuit C is solely composed of phase gadgets, then we can freely rearrange the order of its phase gadgets.

The Hadamard gate gadgetization procedure is presented in Figure 3.2. It results in a $CZ_{a,b}$ gate acting on qubits a and b where a is measured in the Pauli X basis and b is prepared in the $|+\rangle$ state. To undo this Hadamard gate gadgetization, the following two conditions must be satisfied:

1. There must be no gate between the $CZ_{a,b}$ gate and the measurement of qubit a.

2. There must be no gate between the initialization of qubit b and the $CZ_{a,b}$ gate.

The problem of degadgetizing a maximal number of Hadamard gates in a quantum circuit composed of phase gadgets then consists of finding an ordering of its phase gadgets and CZ gates such that these conditions are satisfied for a maximal number of gadgetized Hadamard gates. We can encode these conditions into a directed graph $\mathcal{G}$ where each vertex represents a phase gadget or a CZ gate, and where the arcs represent the precedence constraints between the operations that must be respected for all the Hadamard gates to be degadgetized. Then, the problem consists of finding an ordering of the vertices in $\mathcal{G}$ such that these constraints are satisfied for a maximum number of Hadamard gate gadgetized, which leads to the following problem statement.

Problem 6 (Hadamard gate degadgetization)**.** *Let $\mathcal{P} = \{P^{(1)}, \ldots, P^{(m)}\}$ be a set of Pauli products acting on n qubits such that $P_j^{(i)} \in \{I, Z\}$ for all i, j, and let $\mathcal{H} = \{(a_1, b_1), \ldots, (a_\ell, b_\ell)\}$ be a set of pairs of integers such that $(a_i, b_i) \in \mathbb{Z}_n^2$. Let $\mathcal{G}$ be a directed graph with edge set $V = V_\mathcal{H} \cup V_\mathcal{P}$, where $V_\mathcal{H} = \{h_i \mid (a_i, b_i) \in \mathcal{H}\}$, $V_\mathcal{P} = \{p_i \mid P^{(i)} \in \mathcal{P}\}$, and arc set $A = \{(h_i, h_j) \mid i \neq j, (a_i, b_j) \in \mathcal{H}\} \cup \{(p_i, h_j) \mid P_{a_j}^{(i)} \neq I, P^{(i)} \in \mathcal{P}, (a_j, b_j) \in \mathcal{H}\} \cup \{(h_j, p_i) \mid P_{b_j}^{(i)} \neq I, P^{(i)} \in \mathcal{P}, (a_j, b_j) \in \mathcal{H}\}$. Find a subset $X \subseteq V_\mathcal{H}$ of minimal size such that $\mathcal{G}[V \setminus X]$ is an induced directed acyclic graph.*

10.2 Hadamard gate degadgetization algorithm

Let X be a solution to the Hadamard gate degadgetization problem for a quantum circuit C with diagonal Pauli rotations associated with the Pauli products $\mathcal{P} = \{P^{(1)}, \ldots, P^{(m)}\}$ and with CZ gates induced by the gadgetization of Hadamard gates, which are associated with the pairs of qubits $\mathcal{H} = \{(a_1, b_1), \ldots, (a_\ell, b_\ell)\}$. And let $\mathcal{G}$ be a directed graph with vertex set $V = V_\mathcal{H} \cup V_\mathcal{P}$ constructed from $\mathcal{P}$ and $\mathcal{H}$ as described in the problem statement. A topological sort of the vertices in the induced directed acyclic graph $\mathcal{G}[V \setminus X]$ provides an ordering of the Pauli rotations and CZ gates of the circuit C. This ordering guarantees that the Hadamard gate degadgetization conditions are satisfied for all the Hadamard gates which are not associated with a vertex in X. Therefore, minimizing the number of vertices in X corresponds to maximizing the number of Hadamard gate degadgetized.

The subset of vertices X is called a feedback vertex set of $\mathcal{G}$, as the induced directed graph $\mathcal{G}[V \setminus X]$ is acyclic. Finding a minimum feedback vertex set is a well-known NP-hard problem [169]. Note that we are only considering a subset of all possible feedback vertex sets of $\mathcal{G}$, as X must satisfy $X \subseteq V_\mathcal{H}$. Let $\mathcal{G}'$ be a directed graph with vertex set $V_\mathcal{H}$ and arc set $A' = \{(h_i, h_j) \mid i \neq j, (a_i, b_j) \in \mathcal{H}\} \cup \{(h_i, h_j) \mid (h_i, p_j) \in A, (p_j, h_k) \in A\}$. Notice that for any path in $\mathcal{G}$ between two vertices in $V_\mathcal{H}$ there exists a corresponding path in $\mathcal{G}'$ that traverses the same vertices in $V_\mathcal{H}$. Because there is no arcs between the vertices $V_\mathcal{P} \subset V$ of $\mathcal{G}$, it follows that a feedback vertex set $X \subseteq V_\mathcal{H}$ for $\mathcal{G}'$ is also a feedback vertex set for $\mathcal{G}'$.

To summarize, the complete procedure for degadgetizing a maximal number of Hadamard gates in a quantum circuit C is the following:

1. Construct the graphs $\mathcal{G}$ and $\mathcal{G}'$ associated with C.
2. Find a minimum feedback vertex set X in $\mathcal{G}'$.
3. Construct the circuit associated with a topological sort of the vertices of $\mathcal{G}[V \setminus X]$.

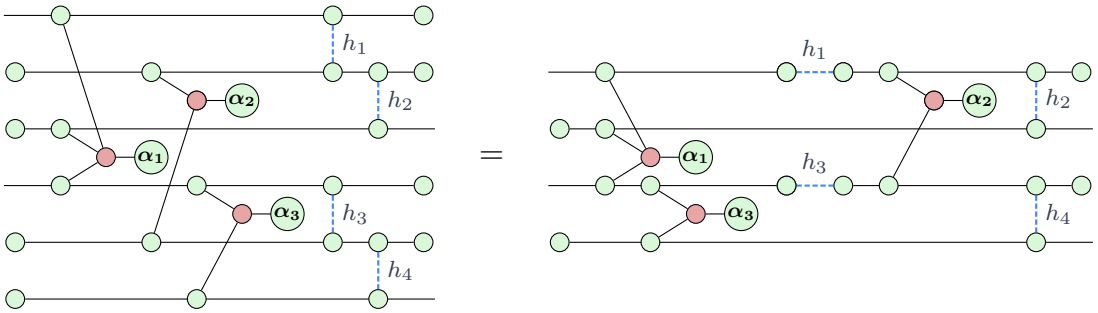
An illustrative example of this procedure is provided in Figure 10.1. Benchmarks for evaluating the performances of this approach are presented in Section 11.3.

10.3 NP-hardness

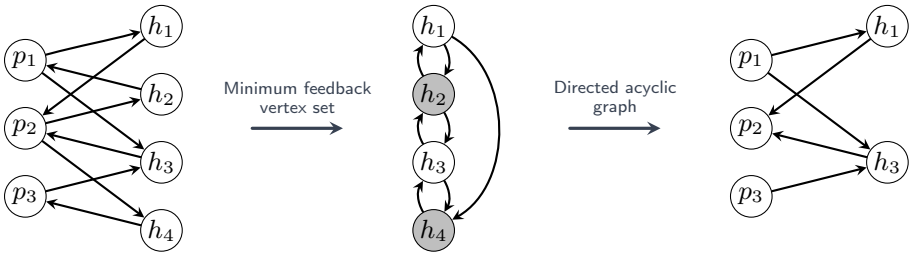
We now prove the NP-hardness of the Hadamard gate degadgetization problem by reducing the minimum directed feedback vertex set problem to this problem.

Theorem 26. *The Hadamard gate degadgetization problem is NP-hard.*

Proof. Let $\mathcal{G}$ be a directed graph with vertex set V and arc set A. Let $\mathcal{P} = \{P^{(u,v)} \mid P^{(u,v)}_{u+|V|} = P^{(u,v)}_v = Z, P^{(u,v)}_w = I, (u,v) \in A, w \notin \{u,v\}\}$ and let $\mathcal{H} = \{(u, u + |V|) \mid u \in V\}$. Let X be a solution to the Hadamard gate degadgetization problem for $\mathcal{P}$ and $\mathcal{H}$. Then, X is a minimum feedback vertex set of the graph $\mathcal{G}'$ with vertex set $V' = \{h_u \mid u \in V\}$ and arc set $A' = \{(h_u, h_v) \mid P^{(u,v)} \in \mathcal{P}\} = \{(h_u, h_v) \mid (u,v) \in A\}$. The directed graph $\mathcal{G}'$ is isomorphic to the directed graph $\mathcal{G}$. Since X is a minimum feedback vertex set for $\mathcal{G}'$, it follows that X is also a feedback vertex set for $\mathcal{G}$. Thus, the Hadamard gate degadgetization problem is at least as hard as the minimum directed feedback vertex set, which is an NP-hard problem [169]. □



(a) The initial quantum circuit (represented in ZX-calculus with phase gadgets) in which Hadamard gates have been gadgetized, and an equivalent circuit in which 2 Hadamard gates have been degadgetized.



(b) Corresponding directed graph where the vertices p_1, p_2, p_3 are associated with the phase gadgets with angle α_1, α_2, α_3 respectively, and the vertices h_1, h_2, h_3, h_4 are associated with the CZ gates. The minimum feedback vertex set, here corresponding to the set of vertices $\{h_2, h_4\}$, indicates the Hadamard gates that will not be degadgetized. A topological sorting of the resulting directed acyclic graph gives the order of the phase gadgets and the Hadamard gates in the optimized circuit.

Figure 10.1: Example demonstrating the procedure used for degadgetizing a maximal number of Hadamard gates in a quantum circuit.

Chapter 11

Qubit-Count Optimization using ZX-Calculus

Besides their fundamental role in quantum computing, qubits are a crucial resource as they can be used as a trade-off for some other resources. The most prominent example is provided by the field of quantum error correction: multiple error-prone qubits can be used to form one reliable logical qubit. Moreover, additional qubits can be valuable for the design and implementation of quantum algorithms as they can sometimes be used to lower the execution time, notably by reducing the computational depth of certain quantum algorithms using various optimization procedures or by enabling their parallelization. Finally, qubits can also be used as a trade-off for quantum gates. For instance, implementing the CCZ gate requires 7 T gates, whereas it can be done with only 4 T gates by incorporating an ancillary qubit [163]. Also, several algorithms designed for optimizing the number of T gates are heavily relying on ancillary qubits [108, 112, 131].

As such, optimizing the number of qubits can enable the execution of a given quantum algorithm if the number of qubits at disposal is insufficient, or it can free up some qubits which can then be used as a trade-off for other resources. In either case, the optimization of the number of qubits is more beneficial if it is not done at the expense of some other critical resources. For instance, non-Clifford gates, which are fundamental for universal quantum computation, are critical resrouces since such gates are costly to implement fault-tolerantly in most quantum error-correcting codes. In this chapter, we present several methods for efficiently optimizing the number of qubits in quantum circuits and in ZX-diagrams depicting lattice surgery operations, without increasing the number of non-Clifford gates. Consequently, the circuits generated by our methods have both an optimized number of non-Clifford gates and an optimized number of qubits, two of the most essential resources for fault-tolerant quantum computing.

Our approach consists of using the ZX-calculus as an intermediate language to establish a connection between our qubit-count optimization problem and well-known graph-theoretical problems. The process then consists of translating a given quantum circuit into a ZX-diagram, performing some transformation on this ZX-diagram and translating it back into a quantum circuit with an optimized number of qubits. This approach naturally includes various methods for optimizing the number of qubits, such as the degadgetization of Hadamard gates, as described above, or qubit reuse with mid-circuit measurements and resets. Qubit reuse consists of finding qubits that can be measured before the end of the circuit, resetting them, and reusing them for the rest of the computation. This approach has already been considered, in particular for near-term quantum computers [170–174]. Our approach differs from these works by also considering both the insertion of new measurements and the removal of some unnecessary measurements (as

done for example with the degadgetization of a Hadamard gate) within the circuit to optimize the number of qubits. These deeper modifications of the circuit are facilitated by the simple and intuitive rules of the ZX-calculus. For example, the gadgetization of a Hadamard gate, as depicted in Figure 3.2, can easily be proved using fundamental rules of the ZX-calculus.

Besides its usage as an intermediate representation for performing optimization in quantum circuits, the ZX-calculus can also be used as a direct representation for other computational models. The Pauli Fusion computational model works natively with the ZX-calculus as it has been designed to reflect its basic structure [75]. Moreover, it has been shown that the generators of the ZX-calculus are closely related to the basic operations of lattice surgery in the surface code [74]. Thus, the Pauli Fusion computational model can be used to represent lattice surgery operations via ZX-diagrams. Our approach can then be used to optimize the number of logical qubits required to implement a given ZX-diagram using lattice surgery operations. However, altering a sequence of lattice surgery operations or modifying a quantum circuit by inserting new measurements can introduce Pauli errors correlated with the measurements outcomes. A method for correcting these errors is then required. This can for example be done by relying on the procedure used in Reference [75] to find a Pauli Fusion flow, which provides a correction strategy for deterministically realizing a sequence of lattice surgery operations.

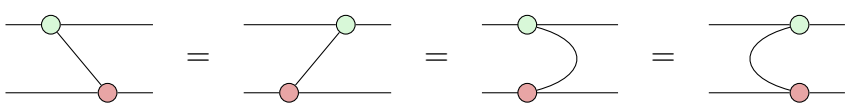
In Section 11.1, we present algorithms for optimizing the number of qubits required to implement a given ZX-diagram using lattice surgery operations. We demonstrate how these results can be used to optimize the number of qubits in quantum circuits in Section 11.2. Benchmarks are provided in Section 11.3 to evaluate the performances of our algorithms on a library of quantum circuits with an optimized number of non-Clifford gates. Finally, in Section 11.4, we propose various perspectives for further research work on this problem based on our results.

11.1 Qubit-count optimization for lattice surgery using ZX-calculus

In this section we present a graph-theoretical approach to the qubit-count optimization problem by relying on the ZX-calculus as a language for lattice surgery. The proposed methods can be used to optimize the number of logical qubits required to implement the lattice surgery operations represented by a given ZX-diagram, without any connectivity constraints. These methods can also be used to optimize the number of qubits in a given quantum circuit, as explained in Section 11.2. In Subsection 11.1.1, we show how we can optimize the number of qubits by rearranging the order of the spiders in the ZX-diagram. Then, in Subsection 11.1.2, we show how we can extend this result by incorporating the spider fusion rule to better optimize the number of qubits.

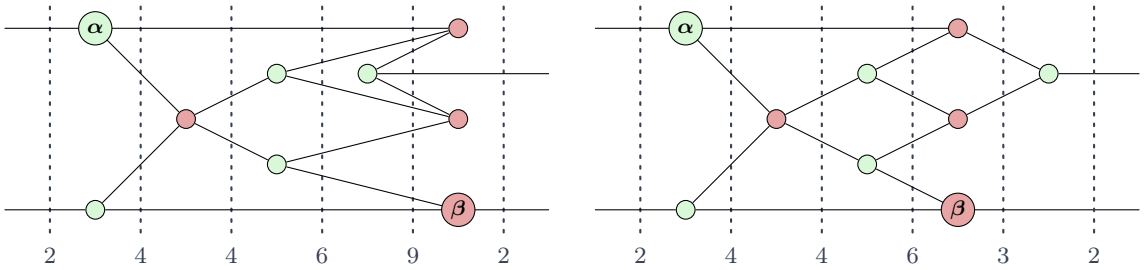
11.1.1 Qubit-count optimization via spider ordering

The operation realized by a ZX-diagram is preserved even when its spiders are moved and its wires are bent: only connectivity matters. For example, the following diagrams all correspond to the CNOT gate, despite representing different sequences of lattice surgery operations:



(11.1)

The number of logical qubits required to implement a given ZX-diagram using lattice surgery operations is equal to the maximum number of wires in any vertical cut of the diagram. Thus, rearranging the order of the spiders of a ZX-diagram or bending (or unbending) its wires can



(a) ZX-diagram whose implementation requires 9 qubits.

(b) Equivalent ZX-diagram, whose implementation requires 6 qubits.

Figure 11.1: Example demonstrating how rearranging the order of spiders in a given ZX-diagrams can reduce the number of logical qubits needed for its implementation. The ZX-diagrams are annotated with the required number of logical qubits at each step to implement them with the lattice surgery operations they represent.

reduce the number of qubits required to implement the associated sequence of lattice surgery operations. An example is provided in Figure 11.1. Then, our objective is to transform a given ZX-diagram by moving its spiders and bending (or unbending) its wires such that the maximum number of wires over any vertical cut of the diagram is minimized. To formalize this problem in a graph-theoretical manner, we define the signature of a ZX-diagram as follows.

Definition 10 (ZX-diagram signature)**.** Let $\mathcal{D}$ be a ZX-diagram, and let $\mathcal{G}_\mathcal{D}$ be the signature graph of $\mathcal{D}$ which is an undirected graph such that

- it contains a vertex for each spider of $\mathcal{D}$, for each input wire of $\mathcal{D}$ ($\mathcal{I}$ denotes this set of vertices) and for each output wire of $\mathcal{D}$ ($\mathcal{O}$ denotes this set of vertices),
- and it has an edge between the vertices u and v for all spiders u and v connected in $\mathcal{D}$, for all $u \in \mathcal{I}$ and spider v incident to the input wire associated with u in $\mathcal{D}$ and for all $u \in \mathcal{O}$ and spider v incident to the output wire associated with u in $\mathcal{D}$.

Then, the triple $(\mathcal{G}_\mathcal{D}, \mathcal{I}, \mathcal{O})$ is the signature associated with $\mathcal{D}$.

And we will use the following definition of vertex ordering for a given graph to incorporate the notion of moving the spiders of a ZX-diagram in our problem.

Definition 11 (Vertex ordering)**.** Let $\mathcal{G}$ be a graph with vertex set V. A function f is an ordering of the vertices V if and only if $f(u) \in \{1, 2, \ldots, |V|\}$ for all $u \in V$ and $f(u) \neq f(v)$ for all $u, v \in V$ such that $u \neq v$.

We will see that our qubit-count optimization problem is closely related to the graph-theoretical concept of cutwidth, defined as follows.

Definition 12 (Cutwidth)**.** Let $\mathcal{G} = (V, E)$ be a graph with vertex set V and edge set E. The cutwidth of a vertex $v \in V$ with respect to a vertex ordering f, denoted by $cw_f(v)$, is defined as:

$$cw_f(v) = |\{(u, w) \mid f(u) \leq f(v) < f(w), (u, w) \in E\}|. \tag{11.2}$$

The cutwidth of $\mathcal{G}$ with respect to f, denoted by $cw_f(\mathcal{G})$, is defined as:

$$cw_f(\mathcal{G}) = \max_{v \in V} cw_f(v). \tag{11.3}$$

And the cutwidth of $\mathcal{G}$, denoted by $cw(\mathcal{G})$, is defined as the minimum $cw_f(\mathcal{G})$ value over all possible orderings:

$$cw(\mathcal{G}) = \min_{f \in S_V} cw_f(\mathcal{G}) = \min_{f \in S_V} \max_{v \in V} cw_f(v) \tag{11.4}$$

where S_V is the set of all orderings of the vertices in V.

The cutwidth problem consists in finding the cutwidth $cw(\mathcal{G})$ of a given graph $\mathcal{G}$. This problem is known to be NP-hard [175].

We now show how rearranging the spiders of a ZX-diagram to minimize the number of qubits required to implement it using lattice surgery operations relates to the cutwidth problem. Let $(\mathcal{G}_\mathcal{D}, \mathcal{I}, \mathcal{O})$ be the signature of a ZX-diagram $\mathcal{D}$ and V be the vertices of $\mathcal{G}_\mathcal{D}$. Choosing an ordering f of the vertices V of $\mathcal{G}_\mathcal{D}$ can be seen as moving the spider of the ZX-diagram $\mathcal{D}$, provided that $f(u) < f(v) < f(w)$ for all $u \in \mathcal{I}$, $v \in V \setminus \{\mathcal{I} \cup \mathcal{O}\}$, and $w \in \mathcal{O}$ (as a spider cannot precede an input wire or succeed an output wire). Indeed, we can construct a ZX-diagram $\mathcal{D}'$ from f such that $\mathcal{D}'$ contains the same nodes as $\mathcal{D}$, connected in the same manner (and connected to the same input and output wires), but where the spiders of $\mathcal{D}'$ are ordered according to f. Then, it follows that the ZX-diagrams $\mathcal{D}'$ and $\mathcal{D}$ are equivalent, but the number of qubits required to implement $\mathcal{D}'$ in lattice surgery is equal to $cw_f(\mathcal{G}_\mathcal{D})$. Thus, we want to find an ordering f such that $cw_f(\mathcal{G}_\mathcal{D})$ is minimized, and where f satisfies $f(u) < f(v) < f(w)$ for all $u \in \mathcal{I}$, $v \in V \setminus \{\mathcal{I} \cup \mathcal{O}\}$, and $w \in \mathcal{O}$. This problem corresponds to the problem of finding an ordering f of the vertices of $\mathcal{G}_\mathcal{D}$ with minimum cutwidth, but where the first and last vertices of the ordering are fixed. We refer to this specific problem as the fixed-endvertices cutwidth problem.

Problem 7 (Fixed-endvertices cutwidth)**.** *Let $\mathcal{G} = (V, E)$ be a graph with vertex set V and edge set E and let $u, w \in V$ such that $u \neq w$. Find an ordering f of the vertices V satisfying $f(u) \leq f(v) \leq f(w)$ for all $v \in V$ and such that $cw_f(\mathcal{G})$ is minimized.*

An illustrative example demonstrating how the fixed-endvertices cutwidth problem can be used to minimize the maximum number of wires in any vertical cut of a given ZX-diagram is provided in Figure 11.2.

We now prove the following theorem, which states that the problem of minimizing the maximum number of wires in any vertical cut of a given ZX-diagram by moving its spiders and bending (or unbending) its wires is equivalent to the fixed-endvertices cutwidth problem.

Theorem 27. *Let $(\mathcal{G}_\mathcal{D}, \mathcal{I}, \mathcal{O})$ be the signature of a given ZX-diagram $\mathcal{D}$. And let $\mathcal{G}'_\mathcal{D}$ be constructed from $\mathcal{G}_\mathcal{D}$ by merging its subset of vertices $\mathcal{I}$ into a single vertex u and its subset of vertices $\mathcal{O}$ into a single vertex w. Then, solving the fixed-endvertices problem for $\mathcal{G}'_\mathcal{D}$ with endvertices u and w is equivalent to solving the problem of finding a ZX-diagram $\mathcal{D}'$ which doesn't have any vertical wire, which can be transformed into $\mathcal{D}$ only by moving its spiders and bending (or unbending) its wires, and such that the maximum number of wires in any vertical cut of $\mathcal{D}'$ is minimized.*

Proof. We prove the equivalence between these two problems by demonstrating that an optimal solution to either problem provides an equivalent solution to the other problem.

Let $f_{u,w}$ be a solution to the fixed-endvertices cutwidth problem for $\mathcal{G}'_\mathcal{D}$ where u and w are the fixed first and last vertices of the ordering $f_{u,w}$. And let $\mathcal{D}'$ be a ZX-diagram constructed from $\mathcal{D}$ by reordering its spiders according to the ordering $f_{u,w}$ and by straightening its wires. Since the spiders of $\mathcal{D}'$ are ordered in the same way as the vertices in $\mathcal{G}'_\mathcal{D}$ with respect to $f_{u,w}$, it follows that the maximum number of wires in any vertical cut of $\mathcal{D}'$ is equal to $cw_{f_{u,w}}(\mathcal{G}'_\mathcal{D})$.

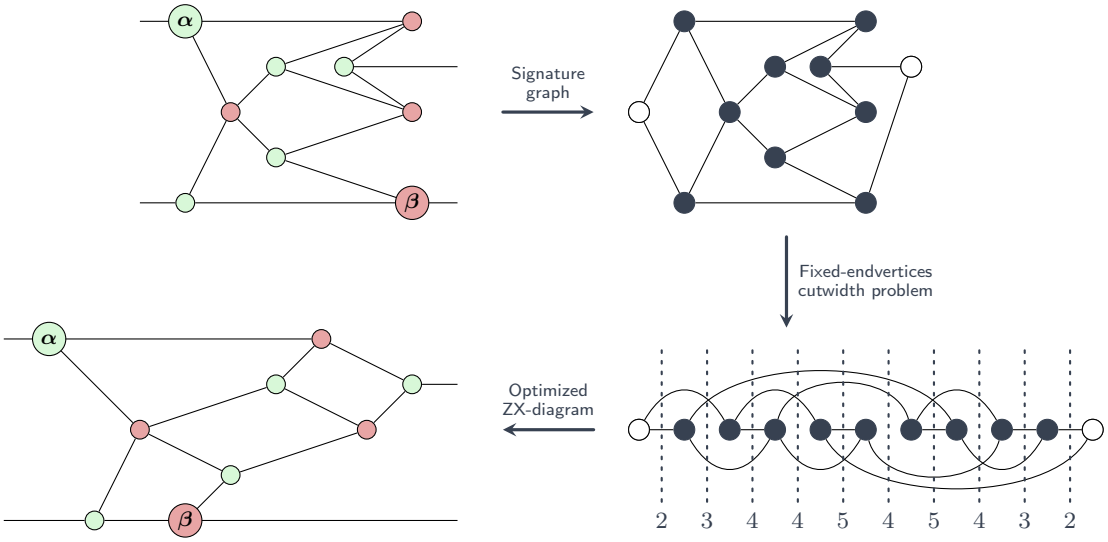


Figure 11.2: Example demonstrating the procedure used for finding an optimal arrangement of the spiders within a given ZX-diagram to minimize the number of logical qubits required for its implementation via lattice surgery operations. We first compute the signature graph $\mathcal{G}_{\mathcal{D}}$ associated with the ZX-diagram. Then, we solve the fixed-endvertices cutwidth problem for $\mathcal{G}_{\mathcal{D}}$. Finally, we rearrange the order of the spiders in the ZX-diagram based on the ordering obtained. We obtain an equivalent ZX-diagram which can be implemented using 5 logical qubits instead of 9 for the initial ZX-diagram.

Let $\mathcal{D}'$ be a ZX-diagram which doesn't have any vertical wire and which can be transformed into $\mathcal{D}$ only by moving its spiders and bending (or unbending) its wires, and such that the maximum number of wires in any vertical cut of $\mathcal{D}'$, noted κ, is minimized. We first show that we can modify $\mathcal{D}'$ to obtain a strict total order of its spiders without increasing the maximum number of wires in any vertical cut of the diagram. Let $U = \{u_1, \ldots, u_n\}$ be a set of spiders in $\mathcal{D}'$ that are vertically aligned. And let $\alpha \leq \kappa$ be the number of wires in the vertical cut of $\mathcal{D}'$ positioned just before the spiders in U, and $\beta \leq \kappa$ be the number of wires in the vertical cut of $\mathcal{D}'$ positioned just after the spiders in U. Let $u_i \in U$ be the spider such that its number of input or output wires is both lower or equal to the number of input wires of every other spider in U and the number of output wires of every other spider in U. If the number of output wires of u_i is lower than its number of input wires, then we modify the diagram $\mathcal{D}'$ by placing the spider u_i just before the set of spiders $U \setminus \{u_i\}$. Otherwise, we modify the diagram $\mathcal{D}'$ by placing the spider u_i just after the set of spiders $U \setminus \{u_i\}$. Let γ be the number of wires in the vertical cut positioned between the spider u_i and the spiders $U \setminus \{u_i\}$. If u_i is placed before the set of spiders $U \setminus \{u_i\}$, then we have $\gamma \leq \alpha$, which implies that $\gamma \leq \kappa$. Otherwise, if u_i is placed after the set of spiders $U \setminus \{u_i\}$, then we have $\gamma \leq \beta$, which also implies that $\gamma \leq \kappa$. The number of wires in all the other vertical cuts of the diagram remains the same. Then, we can repeat this process for every set of vertically aligned spiders in $\mathcal{D}'$ to obtain an equivalent ZX-diagram $\tilde{\mathcal{D}}'$ in which no spiders are vertically aligned, and such that the maximum number of wires in any vertical cut of the diagram is equal to κ. Moreover, note that the ZX-diagram obtained by straightening the wires of $\tilde{\mathcal{D}}'$ also has a maximum number of wires in any vertical cut of the diagram equal to κ. Let $f'_{u,w}$ be the ordering of the vertices in $\mathcal{G}'_{\mathcal{D}}$ corresponding to the ordering of the spiders in $\tilde{\mathcal{D}}'$, then we have $cw_{f'_{u,w}}(\mathcal{G}'_{\mathcal{D}}) = \kappa$. □

The close relation between the cutwidth problem and the fixed-endvertices cutwidth problem can be leveraged to prove the NP-hardness of the fixed-endvertices cutwidth problem as follows.

Theorem 28. *The fixed-endvertices cutwidth problem is NP-hard.*

Proof. Let $\mathcal{G} = (V, E)$ be a graph with vertex set V and edge set E. And let $f_{u,w}$ be an optimal solution to the fixed-endvertices for $\mathcal{G}$ where u and w are the fixed first and last vertices of the ordering $f_{u,w}$. Then, by definition, the cutwidth of $\mathcal{G}$ satisfies

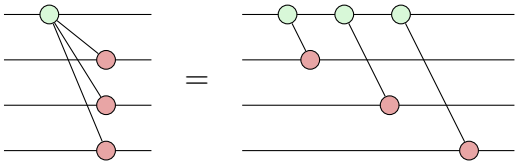
$$cw(\mathcal{G}) = \min_{\substack{u,w \in V \\ u \neq w}} cw_{f_{u,w}}(\mathcal{G}). \tag{11.5}$$

Therefore, we can solve the cutwidth problem by solving the fixed-endvertices cutwidth problem a polynomial number of times, and then use these solutions to compute the cutwidth of $\mathcal{G}$ via Equation 11.5 in polynomial time. Thus, the fixed-endvertices cutwidth problem is at least as hard as the cutwidth problem, which is an NP-hard problem [175]. □

From Theorem 27 and Theorem 28, it follows that the problem of minimizing the number of logical qubits required to implement a given ZX-diagram using lattice surgery operations by moving its spiders and bending (or unbending) its wires is an NP-hard problem. In order to solve this problem more efficiently, it can be useful to first apply some transformation of the initial ZX-diagram to minimize its number of spiders, for instance by using the spider fusion rule presented in Section 3.2.3. Also, spiders having only two incident wires don't have any impact on the optimization problem. Each one of these spiders can be replaced by a single wire and inserted back on their associated wire in the optimized ZX-diagram without altering the obtained solution.

11.1.2 Qubit-count optimization using the fusion rule

The spider fusion rule, presented in Section 3.2.3, can enable significant reductions in the number of qubits required to implement a given ZX-diagram through lattice surgery operations. The most basic example is for implementing a multi-target CNOT gate in lattice surgery, without connectivity constraints. It can either be done with a constant depth but at a linear cost in the number of logical qubits, or, at the extreme opposite, it can be done with only one intermediate logical qubit but at the expense of a linear depth with respect to the number of target qubits:



(11.6)

A simple approach for exploiting this kind of optimization could be to use the spider fusion rule to replace each spider of the diagram by a chain of spiders, each incident to three wires, and then use the results of Section 11.1.1 to optimize the number of qubits. However, the number of ways to unfuse a spider in this manner grows exponentially with respect to the number of wires connected to it, which makes this approach impractical. In this section, we show how the spider fusion rule can be better incorporated into our qubit-count optimization problem and how it relates to the problem of finding the pathwidth of a graph. We first define the pathwidth as follows.

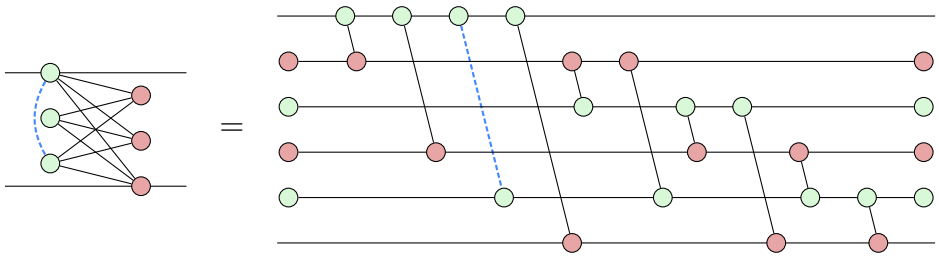


Figure 11.3: Example demonstrating how the maximum number of wires in any vertical cut of a given ZX-diagram can be optimized so that it is equal to $n+1$, where n is the number of spiders in the initial ZX-diagram. One horizontal line is associated with each spider of the initial ZX-diagram. Then, the fusion rule is applied to ensure that any vertical cut of the ZX-diagram is intersecting at most a single wire connecting two spiders in the initial ZX-diagram.

Definition 13 (Pathwidth)**.** Let $\mathcal{G} = (V, E)$ be a graph with vertex set V and edge set E. The pathwidth of a vertex $v \in V$ with respect to a vertex ordering f, denoted by $pw_f(v)$, is defined as:

$$pw_f(v) = |\{u \mid f(u) \leq f(v) < f(w), (u, w) \in E\}|. \tag{11.7}$$

The pathwidth of $\mathcal{G}$ with respect to f, denoted by $pw_f(\mathcal{G})$, is defined as:

$$pw_f(\mathcal{G}) = \max_{v \in V} pw_f(v). \tag{11.8}$$

And the pathwidth of $\mathcal{G}$, denoted by $pw(\mathcal{G})$, is defined as the minimum $pw_f(\mathcal{G})$ value over all possible orderings:

$$pw(\mathcal{G}) = \min_{f \in S_V} pw_f(\mathcal{G}) = \min_{f} \max_{v \in V} pw_f(v) \tag{11.9}$$

where S_V is the set of all orderings of the vertices in V.

This definition of pathwidth was initially referred to as the vertex separation number, before its equivalence to the pathwidth was shown [176]. The pathwidth problem consists in finding the pathwidth $pw(\mathcal{G})$ of a given graph $\mathcal{G}$. This problem is known to be NP-hard [177, 178].

We now show how rearranging the order of the spiders of a ZX-diagram and using the spider fusion rule to minimize the number of qubits required to implement it through surgery operations relates to the pathwidth problem. First, we can notice that the maximum number of wires in any vertical cut of a given ZX-diagram can be optimized so that it is always equal to $n+1$, where n is the number of spiders in the initial ZX-diagram. To do so, we assign an horizontal line to each spider of the ZX-diagram, resulting in n distinct horizontal lines. Then, for each spider u connected to spiders $v_1, \ldots, v_k$, we can use the fusion rule to decompose u it into a sequence of spiders $u_1, \ldots, u_k$ where u_i is connected to v_i and u_i lies on the horizontal line associated with u. This can be done such that the resulting ZX-diagram has a circuit-like form, where the horizontal wires are corresponding to qubits and non-horizontal wires are corresponding to gates acting on these qubits and are performed sequentially. As a result, each vertical cut of the transformed ZX-diagram intersects n horizontal wires and at most one non-horizontal wire. An illustrative example is provided in Figure 11.3.

To further optimize the ZX-diagram, instead of associating each horizontal line with a single spider, we can place several spiders on the same horizontal line if certain conditions are met. We will see that these conditions are closely related to the graph-theoretical notion of path-decomposition, defined as follows [179].

Definition 14 (Path-decomposition)**.** Let $\mathcal{G} = (V, E)$ be a graph with vertex set V and edge set E. A path-decomposition $\mathcal{G}$ is a sequence $X_1, \ldots, X_\ell$ of subsets of V such that:

- for each edge $(u, v) \in E$, there exists i such that $u \in X_i$ and $v \in X_i$;
- for all i, j, k such that $1 \leq i \leq j \leq k \leq \ell$, $X_i \cap X_k \subseteq X_j$.

Let $(\mathcal{G}_\mathcal{D}, \mathcal{I}, \mathcal{O})$ be the signature of a ZX-diagram $\mathcal{D}$, and let $X_1, \ldots, X_\ell$ be a path-decomposition of $\mathcal{G}_\mathcal{D}$ such that $u \in X_1$ for all $u \in \mathcal{I}$ and $u \in X_\ell$ for all $u \in \mathcal{O}$. Then we can construct a ZX-diagram $\mathcal{D}'$, equivalent to $\mathcal{D}$, from the path-decomposition $X_1, \ldots, X_\ell$ which contains ℓ layers of spiders as follows. First, for every spider u in $\mathcal{D}$ we unfuse it in $\mathcal{D}'$ for every layer i where $u \in X_i$. Then, for every wire connecting two spiders u and v in $\mathcal{D}$ we connect these two spiders in a layer i of $\mathcal{D}'$ where $u \in X_i$ and $v \in X_i$. Note that, by definition, the existence of a X_i in the the path-decomposition is guaranteed. Then, similarly to Figure 11.3, we obtain a ZX-diagram which has a circuit-like form, where the wires resulting from the application of the spider fusion rule are horizontal and are corresponding to qubits, and all the other non-horizontal wires are corresponding to gates that can be realized sequentially. In fact, the example of Figure 11.3 corresponds to the trivial case where $\ell = 1$ and all spiders are in X_1. Any vertical cut of $\mathcal{D}'$ intersects at most $\max_i |X_i|$ horizontal wires and at most 1 non-horizontal wire. Therefore, $\mathcal{D}'$ can be implemented trough lattice surgery operations by using at most $\max_i |X_i| + 1$ logical qubits. Thus, we want to find a path-decomposition $X_1, \ldots, X_\ell$ of $\mathcal{G}_\mathcal{D}$ such that $\max_i |X_i|$ is minimized, and where $u \in X_1$ for all $u \in \mathcal{I}$ and $u \in X_\ell$ for all $u \in \mathcal{O}$. The problem of finding a path-decomposition $X_1, \ldots, X_\ell$ of a graph $\mathcal{G}$ that minimizes $\max_i |X_i|$ is equivalent to the problem of finding an ordering f of its vertices such that $pw_f(\mathcal{G})$ is minimized [176]. Therefore, our problem corresponds to the problem of finding an ordering f of the vertices of $\mathcal{G}_\mathcal{D}$ with minimum pathwidth $pw_f(\mathcal{G}_\mathcal{D})$, but where the first and last set of vertices of the ordering f are fixed such that $f(u) < f(v) < f(w)$ for all $u \in \mathcal{I}$, $v \in V \setminus \{\mathcal{I}, \mathcal{O}\}$, $w \in \mathcal{O}$, where V is the vertex set of $\mathcal{G}_\mathcal{D}$. We refer to this specific problem as the fixed-endbags pathwidth problem.

Problem 8 (Fixed-endbags pathwidth)**.** *Let $\mathcal{G} = (V, E)$ be a graph with vertex set V and edge set E and let $U, W \subseteq V$. Find an ordering f of the vertices V satisfying $f(u) < f(v) < f(w)$ for all $u \in U$, $v \in V \setminus \{U, W\}$, $w \in W$ and such that $pw_f(\mathcal{G})$ is minimized.*

Let $(\mathcal{G}_\mathcal{D}, \mathcal{I}, \mathcal{O})$ be the signature of a ZX-diagram $\mathcal{D}$. We will refer to the pathwidth of $\mathcal{D}$, noted $pw(\mathcal{D})$, as the value $pw_f(\mathcal{G}_\mathcal{D})$ such that f is solving the fixed-endbags pathwidth problem where $\mathcal{I}$ and $\mathcal{O}$ are the fixed first and last bags respectively. We now prove the following theorem, which states that a ZX-diagram $\mathcal{D}$ cannot be implemented through lattice surgery operations with less than $pw(\mathcal{D})$ logical qubits if $\mathcal{D}$ can only be transformed by using the spider fusion rule, by moving its spiders and by bending its wires.

Theorem 29. *Let $\mathcal{D}$ be a ZX-diagram in which no spider of the same color are connected by a wire. And let $\mathcal{D}'$ be a ZX-diagram which can be transformed into $\mathcal{D}$ by using the spider fusion rule, by moving its spiders and by bending its wires. Then, the maximum number of wire in any vertical cut of $\mathcal{D}'$ is at least $pw(\mathcal{D})$, i.e. at least $pw(\mathcal{D})$ logical qubits are required to implement $\mathcal{D}'$ through lattice surgery operations.*

Proof. Let $\mathcal{D}'$ be a ZX-diagram which can be transformed into $\mathcal{D}$ by using the spider fusion rule, by moving its spiders and by bending its wires; and let κ be the maximum number of wires in any vertical cut of $\mathcal{D}'$. As demonstrated in the proof of Theorem 27, $\mathcal{D}'$ can be transformed into an equivalent ZX-diagram $\tilde{\mathcal{D}}'$ such that the spiders of $\tilde{\mathcal{D}}'$ have a strict total order (i.e. no two

Algorithm 15: Optimize a ZX-diagram using the fixed-endbags pathwidth problem

Input: A ZX-diagram $\mathcal{D}$.
Output: An optimized ZX-diagram $\mathcal{D}'$ equivalent to $\mathcal{D}$.

$\mathcal{D} \leftarrow \mathcal{D}$ where each pair of connected Z spiders and X spiders have been fused
$(\mathcal{G}_\mathcal{D}, \mathcal{I}, \mathcal{O}) \leftarrow$ signature of $\mathcal{D}$
$f \leftarrow$ solution to the fixed-endbags pathwidth problem for $\mathcal{G}_\mathcal{D}$ with endbags $\mathcal{I}$ and $\mathcal{O}$
$\mathcal{D}' \leftarrow$ copy of $\mathcal{D}$
foreach spider u of $\mathcal{D}$ connected to $n > 1$ spiders $v_1, \ldots, v_n$ such that $f(v_i) < f(v_{i+1})$ **do**
    $\mathcal{D}' \leftarrow$ copy of $\mathcal{D}'$ where u is unfused into n spiders $u_1, \ldots, u_n$ such that u_i is connected to u_{i+1} and v_i
    $f \leftarrow$ copy of f where $f(u_i) = f(u)$
end
$\mathcal{D}' \leftarrow$ copy of $\mathcal{D}'$ where each spider u appears before another spider w if and only if $f(u) < f(w)$ or $f(u) = f(w)$ and $f(u)$ is connected to a spider v such that $f(u) < f(v)$
return $\mathcal{D}'$ with all its wires straightened

spiders in $\tilde{\mathcal{D}}'$ are vertically aligned) and such that the maximum number of wires in any vertical cut of $\tilde{\mathcal{D}}'$ is also equal to κ.

Let $\mathcal{G}_\mathcal{D}$ be the signature graph of $\mathcal{D}$, $\mathcal{G}_{\tilde{\mathcal{D}}'}$ be the signature graph of $\tilde{\mathcal{D}}'$, and let f be the ordering of the vertices in $\mathcal{G}_{\tilde{\mathcal{D}}'}$ corresponding to the ordering of the spiders in $\tilde{\mathcal{D}}'$. Then we have $cw_f(\mathcal{G}_{\tilde{\mathcal{D}}'}) = \kappa$. The cutwidth is always lower or equal to the pathwidth, therefore we have $pw(\tilde{\mathcal{D}}') \leq pw_f(\mathcal{G}_{\tilde{\mathcal{D}}'}) \leq cw_f(\mathcal{G}_{\tilde{\mathcal{D}}'}) = \kappa$. Moreover, the graph $\mathcal{G}_\mathcal{D}$ can be obtain from $\mathcal{G}_{\tilde{\mathcal{D}}'}$ by performing a sequence of edge contraction operations. Performing an edge contraction on a graph does not increase its pathwidth. Therefore, we have $pw(\mathcal{D}) = pw(\mathcal{G}_\mathcal{D}) \leq pw(\mathcal{G}_{\tilde{\mathcal{D}}'}) = pw(\tilde{\mathcal{D}}')$. It follows that $pw(\mathcal{D}) \leq pw(\tilde{\mathcal{D}}') \leq pw_f(\mathcal{G}_{\tilde{\mathcal{D}}'}) \leq cw_f(\mathcal{G}_{\tilde{\mathcal{D}}'}) = \kappa$. □

Consider the algorithm whose pseudo-code is given in Algorithm 15 and which takes a ZX-diagram $\mathcal{D}$ as input. The algorithm starts by transforming $\mathcal{D}$ to minimize its number of spiders using the fusion rule. This transformation ensures that the resulting ZX-diagram doesn't contain any pair of Z spiders or X spiders connected by a wire. Then, the algorithm computes a solution f to the fixed-endbags pathwidth problem for $\mathcal{G}_\mathcal{D}$ with endbags $\mathcal{I}$ and $\mathcal{O}$, where $(\mathcal{G}_\mathcal{D}, \mathcal{I}, \mathcal{O})$ is the signature of $\mathcal{D}$. The ordering f is then used to construct an optimized ZX-diagram $\mathcal{D}'$ equivalent to $\mathcal{D}$. To do so, we can first notice that each vertex u in $\mathcal{G}_\mathcal{D}$ can be associated with an interval that spans from $f(u)$ to the maximum value $f(v)$ where v is a neighbor of u in $\mathcal{G}_\mathcal{D}$ or $v = u$. By constructing the intersection graph of these intervals, we obtain an interval supergraph of $\mathcal{G}$ in which the clique number minus one is equal to $pw_f(\mathcal{G}_\mathcal{D})$ [180]. Each interval represents a space in which the associated spider in $\mathcal{D}$ can be unfused. Then, we obtain a ZX-diagram which has a circuit-like form where the wires resulting from the application of the spider fusion rule are horizontal and are corresponding to qubits, and all the other non-horizontal wires are corresponding to gates that can be realized sequentially. An illustrative example of the complete procedure is provided in Figure 11.4.

The following theorem provides an upper bound on the maximum number of wires in any vertical cut of the ZX-diagram $\mathcal{D}'$ produced by Algorithm 15.

Theorem 30. *Let $\mathcal{D}$ be a ZX-diagram with signature graph $\mathcal{G}_D$, and let $\mathcal{D}'$ be the ZX-diagram produced by Algorithm 15. Then any vertical cut of $\mathcal{D}'$ intersects at most $pw(\mathcal{D}) + 2$ wires, i.e. $\mathcal{D}'$ can be implemented with at most $pw(\mathcal{D}) + 2$ logical qubits through lattice surgery operations.*

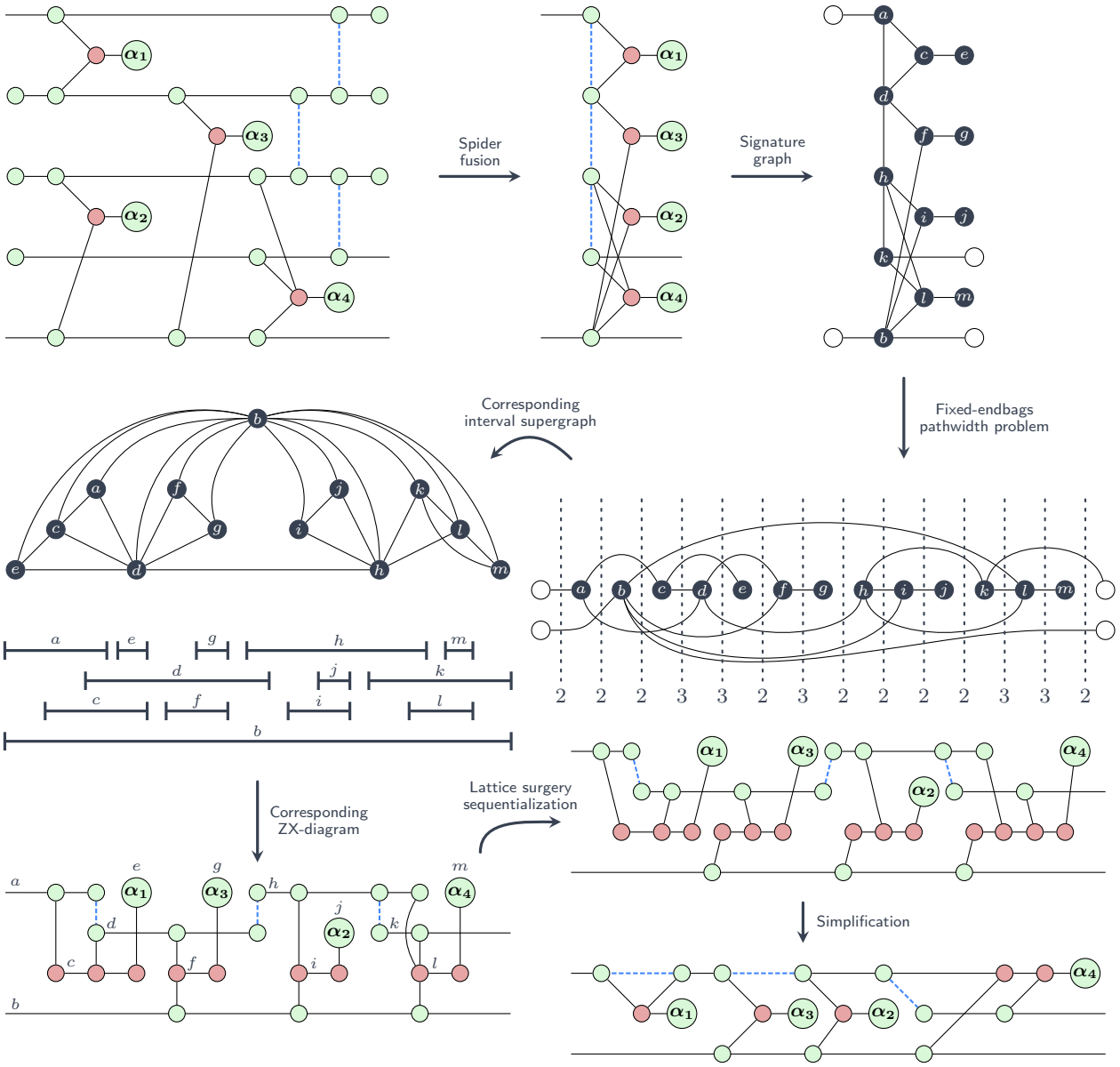
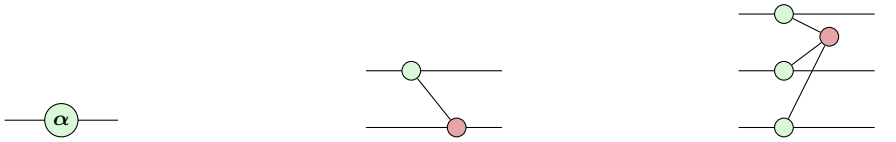


Figure 11.4: Example demonstrating the procedure used for optimizing the number of qubits required to implement a given ZX-diagram using lattice surgery operations by rearranging its spiders and using the spider fusion rule. We first fuse the spiders and compute the signature graph $\mathcal{G}_\mathcal{D}$ associated with the resulting ZX-diagram $\mathcal{D}$. Then, we solve the fixed-endbags pathwidth problem for $\mathcal{G}_\mathcal{D}$. The solution gives us an interval graph where each interval corresponds to a space in which the associated spider of $\mathcal{D}$ must lie and can be unfused. We construct the associated ZX-diagram in which the horizontal wires (except the input and output wires) result from the application of the spider fusion rule, and each vertical wire corresponds to a wire in $\mathcal{D}$. The lattice surgery operations associated with the vertical wires can then be done sequentially, such that there is no split or merge operation involving more than 2 logical qubits simultaneously and more than one non-horizontal wire. Finally, we can simplify the diagram by applying the identity rule and performing parallel operations without increasing the number of wires in any vertical cut of the diagram. We obtain an optimized ZX-diagram (equivalent to $\mathcal{D}$) describing a sequence of lattice surgery operations that can be performed using 4 logical qubits.



(a) $pw(\mathcal{D})$ (b) $pw(\mathcal{D})+1$ (c) $pw(\mathcal{D})+2$

Figure 11.5: Example of ZX-diagrams that are requiring $pw(\mathcal{D})$, $pw(\mathcal{D})+1$ and $pw(\mathcal{D})+2$ logical qubits respectively to be implemented through lattice surgery operations.

Proof. Let f be a solution to the fixed-endbags pathwidth problem for $\mathcal{G}_\mathcal{D}$ with endbags $\mathcal{I}$ and $\mathcal{O}$, where $(\mathcal{G}_\mathcal{D}, \mathcal{I}, \mathcal{O})$ is the signature of $\mathcal{D}$. And let S be the set of intervals defined as follows:

$$S = \{[f(u), f(v)] \mid u \in V, f(v) = \max_{w \in N[u]} f(w)\} \tag{11.10}$$

where V is the vertex set of $\mathcal{G}_\mathcal{D}$ and $N[u]$ denotes the closed neighborhood of the vertex u in $\mathcal{G}_\mathcal{D}$. The intervals in S can be arranged on $pw_f(\mathcal{G}_\mathcal{D}) + 1 = pw(\mathcal{D}) + 1$ distinct horizontal lines such that there is no overlapping point between two intervals on any line [180]. By construction, all the spiders of $\mathcal{D}'$ are positioned on these intervals. Therefore, any vertical cut of $\mathcal{D}'$ intersects at most $pw(\mathcal{D}) + 1$ horizontal wires. Moreover, any vertical cut of $\mathcal{D}'$ intersects at most one non-horizontal wire. Thus, any vertical cut of $\mathcal{D}'$ intersects at most $pw(\mathcal{D}) + 2$ wires. □

Let $\mathcal{D}$ be a ZX-diagram and let $\mathcal{D}'$ be another ZX-diagram which can be transformed into $\mathcal{D}$ by using the spider fusion rule, by moving its spiders and by bending its wires. Theorem 29 gives us a lower bound of $pw(\mathcal{D})$ and Theorem 30 gives us an upper bound of $pw(\mathcal{D}) + 2$ for the number of logical qubits required to implement $\mathcal{D}'$ through lattice surgery operations. These upper and lower bounds are tight. In Figure 11.5, we provide three ZX-diagrams for which the associated optimized ZX-diagram $\mathcal{D}'$ requires at least $pw(\mathcal{D})$, $pw(\mathcal{D}) + 1$, and $pw(\mathcal{D}) + 2$ logical qubits respectively to be implemented through lattice surgery operations.

We now prove the NP-hardness of the fixed-endbags pathwidth problem by relying on its relation with the pathwidth problem.

Theorem 31. *The fixed-endbags pathwidth problem is NP-hard.*

Proof. Let $\mathcal{G} = (V, E)$ be a graph with vertex set V and edge set E. And let $f_{U,W}$ be an optimal solution to the fixed-endbags for $\mathcal{G}$ where U and W are the fixed first and last bags of the ordering $f_{U,W}$. Then, by definition, the pathwidth of $\mathcal{G}$ satisfies

$$pw(\mathcal{G}) = \min_{\substack{u,w \in V \\ u \neq w}} pw_{f_{\{u\},\{w\}}}(\mathcal{G}). \tag{11.11}$$

Therefore, we can solve the pathwidth problem by solving the fixed-endbags pathwidth problem a polynomial number of times, and then use these solutions to compute the pathwidth of $\mathcal{G}$ via Equation 11.11 in polynomial time. Thus, the fixed-endbags pathwidth problem is at least as hard as the pathwidth problem, which is an NP-hard problem [177, 178]. □

Some modifications to the initial ZX-diagram and its associated signature graph can be realized in order to solve the fixed-endbags pathwidth problem more efficiently. Similarly to the fixed-endvertices cutwidth problem, spiders incident to two wires can be replaced by a single wire and inserted back on their associated wire in the optimized ZX-diagram without altering the obtained solution. Also, if a vertex u in a signature graph $\mathcal{G}_\mathcal{D}$ is connected to exactly one vertex v associated with an input or output wire in the ZX-diagram $\mathcal{D}$, then we can remove v from $\mathcal{G}_\mathcal{D}$ and replace it with u in the input or output vertex set.

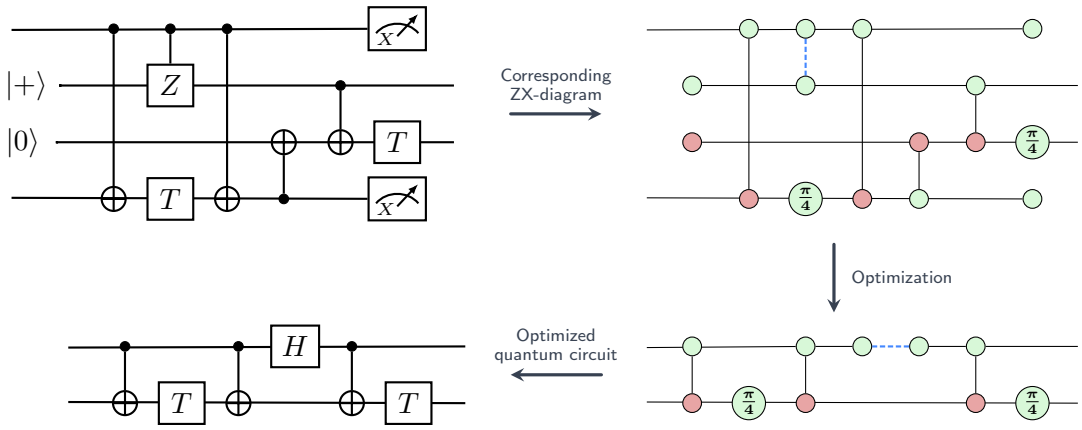


Figure 11.6: Overview of the process described for optimizing the number of qubits in a quantum circuit. The initial quantum circuit is translated into a ZX-diagram. Then, the ZX-diagram is optimized using the method explained in Section 11.1. Finally, the optimized ZX-diagram is translated back into a quantum circuit with an optimized number of qubits.

11.2 Qubit-count optimization in quantum circuits using ZX-calculus

In this section, we rely on the ZX-calculus as an intermediate language to exploit the results of Section 11.1 and optimize the number of qubits in a quantum circuit. The procedure for optimizing the number of qubits in a given quantum circuit C using Algorithm 15 is as follows:

1. Translate C into a ZX-diagram $\mathcal{D}$ by using the translation table of Figure 3.1, and use the spider fusion rule to minimize the number of spiders within $\mathcal{D}$.

2. Execute Algorithm 15 with $\mathcal{D}$ given as input to produce an optimized and equivalent ZX-diagram $\mathcal{D}'$.

3. Translate $\mathcal{D}'$ into a quantum circuit C' by using the translation table of Figure 3.1.

An example is provided in Figure 11.6. Translating a quantum circuit into a ZX-diagram can always easily be done, for example by using the procedure described in Section 3.3.1. The spider fusion rule is applied to minimize the number of spiders within $\mathcal{D}$ in order to improve the effectiveness of Algorithm 15. Note that a more sophisticated procedure, using other rules than the spider fusion rule, could be used here to transform $\mathcal{D}$ before executing Algorithm 15. We leave it as a question for future research to determine which transformation procedure for $\mathcal{D}$ could be used to maximize the effectiveness of Algorithm 15. The last step of the procedure consists of translating the optimized ZX-diagram $\mathcal{D}'$ into a quantum circuit C'. We can notice that a non-horizontal wire in $\mathcal{D}'$ is either connecting a Z spider to an X spider, or is a dashed blue edge connected two Z spiders. In the first case, it can be translated to a CNOT gate; and in the second case, it can be translated to a CZ gate. And all the horizontal wires in $\mathcal{D}$ can be mapped to horizontal wires in C', where they represent qubits. Thus, we obtain a quantum circuit C' which has a number of qubits characterized by the following theorem.

Theorem 32. *Let $\mathcal{D}$ be a ZX-diagram in which no spider of the same color are connected by a wire. Let $\mathcal{D}'$ be a ZX-diagram produced by Algorithm 15 when $\mathcal{D}$ is given as input and let C' be the quantum circuit translated from $\mathcal{D}'$ using the translation table of Figure 3.1. Then the number of qubits in C' is at least equal to $pw(\mathcal{D})$ and at most equal to $pw(\mathcal{D}) + 1$.*

Proof. Each non-horizontal wire in $\mathcal{D}'$ is either connected a Z spider to a X spider or is a dashed blue edge connected two Z spiders. Therefore, each non-horizontal wire is translated as either a CNOT gate or a CZ gate in C'. As stated by Theorem 29, the maximum number of horizontal wires in any vertical cut of $\mathcal{D}'$ is at least equal to $pw(\mathcal{D})$. And as demonstrated in the proof of Theorem 30, the maximum number of horizontal wires in any vertical cut of $\mathcal{D}'$ is at most equal to $pw(\mathcal{D}) + 1$. Each horizontal wire in $\mathcal{D}'$ is translated as a qubit in C'. Therefore, the number of qubits in C' is at least equal to $pw(\mathcal{D})$ and at most equal to $pw(\mathcal{D}) + 1$. □

11.3 Benchmarks

In this section, we evaluate the qubit-count reductions achieved by our approaches on a set of quantum circuits in which the number of T gates has been optimized. To solve the Hadamard gates degadgetization problem, we used the optimal solver of the directed feedback vertex set problem described in Reference [181]. For the fixed-endbags pathwidth problem, we applied a fast but non-optimal greedy algorithm. Our implementations of the algorithms used for the benchmarks are open source [8].

The quantum circuits were obtained from References [127] and [128]. These circuits were pre-optimized as follows. First, the `FastTMerge` algorithm (Algorithm 11) was applied to rapidly and efficiently reduce the number of T gates in the circuits. This method optimally minimizes the number of non-Clifford gates in Clifford+R_Z circuits when no information about the angles of the R_Z gates is known, as proven in Chapter 8. Then, the `InternalHOpt` algorithm of Chapter 6 was applied to optimize the number of internal Hadamard gates in the circuits. This method is also optimal for Clifford+R_Z circuits when no information about the angles of the R_Z gates is known, as proven in Chapter 8. Finally, the Hadamard gates of the circuits were gadgetized and the `FastTODD` algorithm of Chapter 9 was applied to optimize the number of T gates in the circuits. We chose the `FastTODD` T-count optimizer of Chapter 9 as it currently yields the best results on most of the quantum circuits, while being faster than other T-count optimizers. Note that the benchmark results may vary depending on the pre-optimization procedure used, such as when a different T-count optimizer is applied.

The benchmark results are presented in Table 11.1. We can notice that the approach based on the fixed-endbags pathwidth problem performs as well as or surpasses the Hadamard gates degadgetization approach on most quantum circuits evaluated. Nevertheless, the Hadamard gates degadgetization approach is performing better than the approach based on the fixed-endbags pathwidth problem on a few quantum circuits. This discrepancy is probably due to the non-optimality of the greedy algorithm used in the benchmarks for the fixed-endbags pathwidth problem. As future work, it would be valuable to develop more efficient algorithms for tackling the fixed-endbags pathwidth problem. An optimal algorithm for the fixed-endbags pathwidth problem would certainly have a large complexity due to the NP-hardness of the problem but could potentially be executed on small circuits.

11.4 Perspectives

This section outlines several potential avenues for future research work based on the results presented herein.

	Pre-optimization			Qubit-count	
Circuit	n	h	Qubit-count	Degadgetization	Pathwidth
Adder_8	24	37	61	58	55
Barenco Tof_3	5	3	8	7	7
Barenco Tof_4	7	7	14	11	10
Barenco Tof_5	9	11	20	13	12
Barenco Tof_{10}	19	31	50	36	31
CSLA MUX_3	15	6	21	20	20
CSUM MUX_9	30	12	42	40	38
Grover_5	9	68	77	57	33
Ham_{15} (high)	20	331	351	175	123
Ham_{15} (low)	17	29	46	35	31
Ham_{15} (med)	17	54	71	46	37
Mod Adder_{1024}	28	304	332	246	198
Mod Mult_{55}	9	3	12	10	12
Mod Red_{21}	11	17	28	24	19
QCLA Adder_{10}	36	25	61	55	49
QCLA Com_7	24	18	42	36	32
QCLA Mod_7	26	58	84	71	41
QFT_4	5	38	43	8	8
RC Adder_6	14	10	24	22	23
Tof_3	5	2	7	6	6
Tof_4	7	4	11	9	10
Tof_5	9	6	15	13	12
Tof_{10}	19	16	35	27	25
VBE Adder_3	10	4	14	13	13

Table 11.1: Qubit-count achieved by different methods on a set of quantum circuits with an optimized number of T gates. The values n and h are respectively corresponding to the number of qubits and the number of internal Hadamard gates gadgetized in the initial circuit. The qubit-count in the initial circuit is equal to $n + h$. The degadgetization column refers to the number of qubits achieved by the Hadamard gates degadgetization method. The pathwidth column refers to the number of qubits achieved by the approach based on the fixed-endbags pathwidth problem. All circuits were processed by the algorithms in no more than a few seconds.

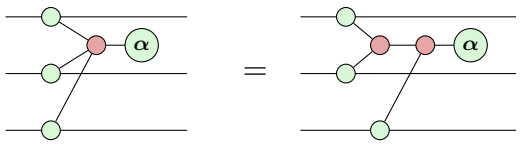
11.4.1 Correction strategy and depth optimization

To deterministically realize an operator in lattice surgery, it is required to have a correction strategy in order to counteract the non-deterministic nature of some lattice surgery operations. This correction strategy imposes a specific order in which the lattice surgery operations must be realized. Therefore, while necessary for determinism, the correction strategy is limiting the actions that can be performed to optimize the number of qubits. A method for finding a correction strategy, called PF-flow, for an arbitrary ZX-diagram has been proposed in Reference [75]. However, the completeness of this approach has not been proved. This means that there may be some ZX-diagrams which don't have a PF-flow but which can still be implemented deterministically trough lattice surgery operations by using another correction strategy. Therefore, an open problem is to find a complete correction strategy, and to better characterize the set of ZX-diagrams which are deterministically implementable using lattice surgery operations. Addressing these challenges could potentially lead to better optimization in the qubit-count.

Furthermore, the PF-flow has been designed to optimize the computational depth, as it aims at finding a correction strategy that divides the ZX-diagram in a minimized number of layers. A simple method for optimizing both the depth and the qubit-count could consists in finding a PF-flow and then applying our methods for optimizing the number of qubits in each layer of the ZX-diagram. However, this approach does not minimize the number of qubits in between two layers of the ZX-diagram. An open problem is to develop a more comprehensive strategy to simultaneously optimize both the computational depth and the number of qubits.

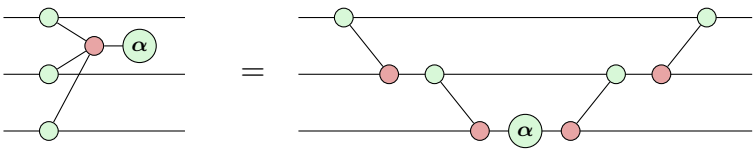
11.4.2 Qubit-count optimization using a complete ZX-calculus equational theory

We proposed an approach to optimize the number of logical qubits required to implement a given ZX-diagram through lattice surgery operations. Our optimization procedure rearranges the order of the spiders and makes use of the identity and fusion rules of the ZX-calculus. A natural extension would be to consider other ZX-calculus rules that could lead to even better optimization. For instance, applying our algorithm to the following ZX-diagram, which represents a phase gadget, yields a ZX-diagram for which 5 logical qubits are required to realize the lattice surgery operations it represents:



(11.12)

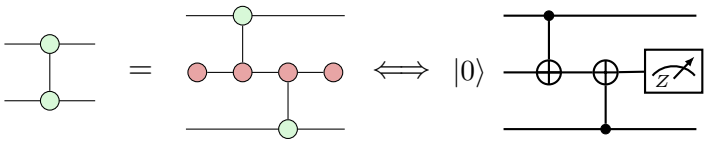
However, as shown below, there exists an equivalent ZX-diagram which can be implemented in lattice surgery with only 4 logical qubits:



(11.13)

Ultimately, our goal is to rely on a complete ZX-calculus equational theory to find an equivalent ZX-diagram which can be implemented in lattice surgery by using a minimal number of qubits. Our results indicates that this consists in finding an equivalent ZX-diagram $\mathcal{D}$ such that $pw(\mathcal{D})$ is minimized. Additionally, we would like to maintain the ability to translate the optimized ZX-diagram directly into a quantum circuit, akin to the straightforward translation

enabled by the proposed approach based on the fixed-endbags pathwidth problem. In our approach, this is guaranteed by the fact that each non-horizontal wire corresponds either to a dashed blue edge connecting two Z spiders, representing a CZ gate, or a wire connected a Z spider and a X spider, representing a CNOT gate. For other cases, an intermediate qubit may be required to translate the ZX-diagram into a quantum circuit, for example:



(11.14)

Conclusion

The primary objective of this thesis was to contribute to the development of a comprehensive and high-performance compilation stack for fault-tolerant quantum computing. Our results span various levels of the compilation stack, addressing critical challenges in quantum circuit synthesis, quantum circuit optimization, and fault-tolerant implementation.

In the domain of quantum circuit synthesis, our work focused on the synthesis of quantum arithmetic operators. Namely, in Chapter 4, we introduced the first ancilla-free exact quantum adder with sublinear depth. Additionally, in Chapter 5, we presented a novel approach for constructing efficient quantum circuits that perform binary field multiplication, a fundamental operation in quantum cryptanalysis of binary elliptic curve cryptography. Our method achieves a Toffoli gate count of $\mathcal{O}(n^{\log_2(3)})$, matching the best known results, while offering a significant improvement in terms of space-time cost. More specifically, the circuits generated by our algorithm have a linear depth and use $\mathcal{O}(n \log_2(n))$ ancillary qubits. This corresponds to a space-time cost of $\mathcal{O}(n^2 \log_2(n))$, which is the best-known space-time cost for quantum circuits implementing binary field multiplication with a subquadratic number of Toffoli gates. An open question is whether this space-time cost can be further reduced. We partially answered this question positively by demonstrating that there are indeed cases where further optimization is possible. In particular, when the multiplication is performed modulo a primitive polynomial which is either a trinomial or an equally spaced polynomial, we showed that a logarithmic depth can be achieved by using $\mathcal{O}(n^{\log_2(3)})$ ancillary qubits. An interesting direction for future work would be to investigate whether there exist other families of primitive polynomials for which multiplication can be performed in logarithmic depth.

The core of the results presented in this thesis focuses on quantum circuit optimization. In Chapter 6, we presented an algorithm to realize the synthesis of a sequence of Pauli rotations over the $\{X, \text{CNOT}, S, H, R_Z\}$ gate set using a minimal number of Hadamard gates and with a time complexity of $\mathcal{O}(n^2 m)$, in the typical case where $n \leq m$, and where n is the number of qubits and m is the number of Pauli rotations. A closely related problem is to optimize a Clifford+R_Z circuit so that the sequence of Pauli rotations it is implementing contains a minimal number of internal Hadamard gates, where a Hadamard gate is called internal if it is comprised between the first and last non-Clifford R_Z gates of the circuit. Solving this problem is important to improve the efficiency and scalability of algorithms minimizing the number of non-Clifford R_Z gates such as T-count optimizers, and to minimize the additional cost that comes with the Hadamard gates gadgetization procedure.

Our algorithms are optimal for a given sequence of Pauli rotations, however there may exist other sequences of Pauli rotations, associated with the same operator, which could be implemented with fewer Hadamard gates. An open problem is to find a sequence of Pauli rotations implementing a given unitary gate up to a Clifford operator such that the rank of the associated commutativity matrix is minimal. Should there exist an algorithm solving this problem in reasonable time, then it could be used in conjunction with our algorithms to implement a unitary

gate over the Clifford+R_Z gate set with a minimal number of internal Hadamard gates.

In Chapter 7, we presented an algorithm to optimize the number of non-Clifford rotation gates in a quantum circuit. Our algorithm has a better complexity than other algorithms having the same optimization strategy and benchmarks show that its execution time is much lower. When executed on a parametrized quantum circuit, we proved in Chapter 8 that our algorithm produces a circuit containing a minimal number of parametrized rotations under the condition that all constant rotations of the circuit are Clifford rotations and that no parameter appears more than once. Under the same conditions, we also proved that the Hadamard gate optimization algorithm presented in Chapter 6 produces a circuit containing a minimal number of Hadamard gates and internal Hadamard gates.

It was first shown in Reference [138] that specific knowledge of the phases involved in the circuit is required to do better at removing non-Clifford rotation gates. Our results corroborate this with some interesting additional insights. In particular, our results hold for any kind of parameter transformations that do not clone parameters. Notably, we showed that allowing parameter transformations that are discontinuous does not enable better optimization. For instance, this imply that the Euler angle transformation is not useful to find an equivalent parametrized Clifford circuit with non-repeated parameters containing a lower number of parametrized rotations. Also, it is interesting to note that specific knowledge of the phases involved in the circuit is required not only to better optimize the number of non-Clifford rotation gates, but also to better optimize the number of Hadamard gates. This further reinforces the interdependence between the non-Clifford rotation gate optimization problem and the Hadamard gate optimization problem, as highlighted in Chapter 6.

An obvious generalization of our results would be to remove one of the two restrictions required to claim optimality. The first one is that all constant rotations must be Clifford rotations. It has been proven that allowing non-Clifford constant rotations leads to an NP-hard problem [138]. Removing the second restriction would consist in allowing the parameters to appear more than once in the circuit. This problem is conjecture to be NP-hard [138]. Due to the projected difficulty of these generalizations, we might consider smaller incremental steps. For instance, we could allow parameters to be repeated only a small number of times. Another generalization could be to allow repeated parameters only for parametrized Clifford rotations. Any progress on these problems would bring us closer to an optimal compilation of parametrized quantum circuits.

In Chapter 9, we presented several polynomial-time algorithms for reducing the number of T gates in a Clifford+T circuit. Benchmarks show that our algorithms consistently achieve the lowest execution times and provide the best T-count reduction on almost all the quantum circuits evaluated when compared to state-of-the-art T-count optimizers. As such, our algorithms not only achieve state-of-the-art T-count reduction but also offer much greater scalability, thereby allowing efficient T-count optimization on larger quantum circuits.

We proved that our algorithms are producing a circuit in which the number of T gates is upper bounded by $(n^2 + n)/2 + 1$ when they are executed on a Hadamard-free circuit, where n is the number of qubits. It has been shown that there exists an asymptotic upper bound of $n^2/2-1$ [182]. The question of whether or not there exists a polynomial-time algorithm satisfying this asymptotic upper bound remains open.

We also demonstrated how the number of T gates in a Clifford+T circuit can be optimized such that it is lower or equal to $(n + 1)(n + 2h)/2 + 1$ where n is the number of qubits and h is the number of internal Hadamard gates in the circuit, without using any ancillary qubits and in polynomial time. This reinforces the interdependence between the problems of optimizing the number of internal Hadamard gates and optimizing the number of T gates. As part of future

work, it would be beneficial to more clearly determine the roles that internal Hadamard gates and ancillary qubits have in the T-count optimization problem.

Finally, in Chapters 10 and 11, we proposed novel methods for optimizing the number of qubits in quantum circuits and the number of logical qubits required to perform lattice surgery operations. On the basis of our results, we suggested several promising avenues for future research work. These include extending the proposed approach to potentially achieve even better reduction in the number of qubits, and investigating how our approach could be adapted to take into account other important metrics such as the computational depth. These results make use of the ZX-calculus and its fundamental rules to seamlessly formulate the qubit-count optimization problem and establish a connection with well-studied problems in graph theory. As such, our findings provide a compelling case for the use of the ZX-calculus in quantum compilation. This further motivates the adoption of the ZX-calculus as a formal, powerful, and versatile tool for representing and optimizing quantum computation.

Personal Bibliography

[1] Maxime Remaud and Vivien Vandaele. Ancilla-free Quantum Adder with Sublinear Depth. *arXiv preprint*, 2025. arXiv:2501.16802.

[2] Vivien Vandaele. Quantum binary field multiplication with subquadratic Toffoli gate count and low space-time cost. *arXiv preprint*, 2025. arXiv:2501.16136.

[3] Vivien Vandaele, Simon Martiel, Simon Perdrix, and Christophe Vuillot. Optimal Hadamard Gate Count for Clifford+T Synthesis of Pauli Rotations Sequences. *ACM Transactions on Quantum Computing*, 5(1):1–29, February 2024. arXiv:2302.07040. doi:10.1145/3639062.

[4] Vivien Vandaele, Simon Perdrix, and Christophe Vuillot. Optimal number of parametrized rotations and Hadamard gates in parametrized Clifford circuits with non-repeated parameters. *arXiv preprint*, 2024. arXiv:2407.07846.

[5] Vivien Vandaele. Lower T-count with faster algorithms. *arXiv preprint*, 2024. arXiv:2407.08695.

[6] Vivien Vandaele. Qubit-count optimization using ZX-calculus. *arXiv preprint*, 2024. arXiv:2407.10171.

[7] Vivien Vandaele, Simon Martiel, and Timothée Goubault de Brugière. Phase polynomials synthesis algorithms for NISQ architectures and beyond. *Quantum Science and Technology*, 7(4):045027, September 2022. arXiv:2104.00934. doi:10.1088/2058-9565/ac5a0e.

[8] https://github.com/VivienVandaele/quantum_circuit_optimization, 2024.

[9] https://github.com/VivienVandaele/quantum-binary-field-multiplier-synthesis, 2024.

Bibliography

[10] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of Statistical Physics*, 22(5):563–591, May 1980. doi:10.1007/bf01011339.

[11] Richard P. Feynman. *Simulating Physics with Computers*, page 133–153. CRC Press, 2002. doi:10.1201/9780429500459-11.

[12] Yuri Manin. Computable and uncomputable. *Sovetskoye Radio, Moscow*, 128:28, 1980.

[13] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439(1907):553–558, 1992. doi:10.1098/rspa.1992.0167.

[14] Daniel R Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997. doi:10.1137/S0097539796298637.

[15] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE Comput. Soc. Press, 1994. doi:10.1109/sfcs.1994.365700.

[16] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing - STOC '96*, STOC '96, page 212–219. ACM Press, 1996. doi:10.1145/237814.237866.

[17] Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5), November 2004. arXiv:quant-ph/0406196. doi:10.1103/physreva.70.052328.

[18] Daniel Gottesman. The Heisenberg representation of quantum computers. *arXiv preprint*, 1998. arXiv:quant-ph/9807006.

[19] A. Kitaev, A. Shen, and M. Vyalyi. *Correspondence between classical and quantum computation*. American Mathematical Society, May 2002. doi:10.1090/gsm/047/10.

[20] CM Dawson and MA Nielsen. The Solovay-Kitaev algorithm. *Quantum Information and Computation*, 6(1):81–95, 2006. arXiv:quant-ph/0505030. doi:10.26421/qic6.1-6.

[21] David Gosset, Vadym Kliuchnikov, Michele Mosca, and Vincent Russo. An algorithm for the T-count. *Quantum Information & Computation*, 14(15-16):1261–1276, November 2014. arXiv:1308.4134. doi:10.26421/qic14.15-16-1.

[22] Cody Jones. Low-overhead constructions for the fault-tolerant Toffoli gate. *Physical Review A*, 87(2), February 2013. arXiv:1212.5069. doi:10.1103/physreva.87.022328.

[23] Austin G Fowler and Craig Gidney. Low overhead quantum computation using lattice surgery. *arXiv preprint*, 2018. arXiv:1808.06709.

[24] Daniel Litinski. A game of surface codes: Large-scale quantum computing with lattice surgery. *Quantum*, 3:128, 2019. arXiv:1808.02892. doi:10.22331/q-2019-03-05-128.

[25] Junhong Nie, Wei Zi, and Xiaoming Sun. Quantum circuit for multi-qubit Toffoli gate with optimal resource. *arXiv preprint*, 2024. arXiv:2402.05053.

[26] Tanuj Khattar and Craig Gidney. Rise of conditionally clean ancillae for optimizing quantum circuits. *arXiv preprint*, 2024. arXiv:2407.17966.

[27] M. Fang, S. Fenner, F. Green, S. Homer, and Y. Zhang. Quantum lower bounds for fanout. *Quantum Information and Computation*, 6(1):46–57, January 2006. arXiv:quant-ph/0312208. doi:10.26421/qic6.1-3.

[28] Anne Broadbent and Elham Kashefi. Parallelizing quantum circuits. *Theoretical Computer Science*, 410(26):2489–2510, June 2009. arXiv:quant-ph/0704.1736. doi:10.1016/j.tcs.2008.12.046.

[29] Yasuhiro Takahashi, Seiichiro Tani, and Noboru Kunihiro. Quantum addition circuits and unbounded fan-out. *Quantum Information and Computation*, 10(9 & 10):872–890, September 2010. arXiv:0910.2530. doi:10.26421/qic10.9-10-12.

[30] Peter W. Shor. Scheme for reducing decoherence in quantum computer memory. *Physical Review A*, 52(4):R2493–R2496, October 1995. doi:10.1103/physreva.52.r2493.

[31] A. M. Steane. Error Correcting Codes in Quantum Theory. *Physical Review Letters*, 77(5):793–797, July 1996. doi:10.1103/physrevlett.77.793.

[32] Daniel Gottesman. Stabilizer Codes and Quantum Error Correction. *PhD thesis, California Institute of Technology*, 1997. arXiv:quant-ph/9705052.

[33] P.W. Shor. Fault-tolerant quantum computation. In *Proceedings of 37th Conference on Foundations of Computer Science*, SFCS-96, page 56–65. IEEE Comput. Soc. Press, 1996. arXiv:quant-ph/9605011. doi:10.1109/sfcs.1996.548464.

[34] D. Aharonov and M. Ben-Or. Fault-tolerant quantum computation with constant error. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing - STOC '97*, STOC '97, page 176–188. ACM Press, 1997. arXiv:quant-ph/9906129. doi:10.1145/258533.258579.

[35] Emanuel Knill, Raymond Laflamme, and Wojciech H. Zurek. Resilient quantum computation: error models and thresholds. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1969):365–384, 1998. arXiv:quant-ph/9702058. doi:10.1098/rspa.1998.0166.

[36] A.Yu. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, January 2003. doi:10.1016/s0003-4916(02)00018-0.

[37] Dominic Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, December 2012. arXiv:1111.4022. doi:10.1088/1367-2630/14/12/123011.

[38] Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Physical Review A*, 71(2), February 2005. arXiv:quant-ph/0403025. doi:10.1103/physreva.71.022316.

[39] Austin G. Fowler. Coping with qubit leakage in topological codes. *Physical Review A*, 88(4), October 2013. arXiv:1308.6642. doi:10.1103/physreva.88.042308.

[40] Zijun Chen, Julian Kelly, Chris Quintana, R. Barends, B. Campbell, Yu Chen, B. Chiaro, A. Dunsworth, A. G. Fowler, E. Lucero, E. Jeffrey, A. Megrant, J. Mutus, M. Neeley, C. Neill, P. J. J. O'Malley, P. Roushan, D. Sank, A. Vainsencher, J. Wenner, T. C. White, A. N. Korotkov, and John M. Martinis. Measuring and Suppressing Quantum State Leakage in a Superconducting Qubit. *Physical Review Letters*, 116(2), January 2016. arXiv:1509.05470. doi:10.1103/physrevlett.116.020501.

[41] Kevin C. Miao, Matt McEwen, Juan Atalaya, Dvir Kafri, Leonid P. Pryadko, Andreas Bengtsson, Alex Opremcak, Kevin J. Satzinger, Zijun Chen, Paul V. Klimov, Chris Quintana, Rajeev Acharya, Kyle Anderson, Markus Ansmann, Frank Arute, Kunal Arya, Abraham Asfaw, Joseph C. Bardin, Alexandre Bourassa, Jenna Bovaird, Leon Brill, Bob B. Buckley, David A. Buell, Tim Burger, Brian Burkett, Nicholas Bushnell, Juan Campero, Ben Chiaro, Roberto Collins, Paul Conner, Alexander L. Crook, Ben Curtin, Dripto M. Debroy, Sean Demura, Andrew Dunsworth, Catherine Erickson, Reza Fatemi, Vinicius S. Ferreira, Leslie Flores Burgos, Ebrahim Forati, Austin G. Fowler, Brooks Foxen, Gonzalo Garcia, William Giang, Craig Gidney, Marissa Giustina, Raja Gosula, Alejandro Grajales Dau, Jonathan A. Gross, Michael C. Hamilton, Sean D. Harrington, Paula Heu, Jeremy Hilton, Markus R. Hoffmann, Sabrina Hong, Trent Huang, Ashley Huff, Justin Iveland, Evan Jeffrey, Zhang Jiang, Cody Jones, Julian Kelly, Seon Kim, Fedor Kostritsa, John Mark Kreikebaum, David Landhuis, Pavel Laptev, Lily Laws, Kenny Lee, Brian J. Lester, Alexander T. Lill, Wayne Liu, Aditya Locharla, Erik Lucero, Steven Martin, Anthony Megrant, Xiao Mi, Shirin Montazeri, Alexis Morvan, Ofer Naaman, Matthew Neeley, Charles Neill, Ani Nersisyan, Michael Newman, Jiun How Ng, Anthony Nguyen, Murray Nguyen, Rebecca Potter, Charles Rocque, Pedram Roushan, Kannan Sankaragomathi, Henry F. Schurkus, Christopher Schuster, Michael J. Shearn, Aaron Shorter, Noah Shutty, Vladimir Shvarts, Jindra Skruzny, W. Clarke Smith, George Sterling, Marco Szalay, Douglas Thor, Alfredo Torres, Theodore White, Bryan W. K. Woo, Z. Jamie Yao, Ping Yeh, Juhwan Yoo, Grayson Young, Adam Zalcman, Ningfeng Zhu, Nicholas Zobrist, Hartmut Neven, Vadim Smelyanskiy, Andre Petukhov, Alexander N. Korotkov, Daniel Sank, and Yu Chen. Overcoming leakage in quantum error correction. *Nature Physics*, 19(12):1780–1786, October 2023. arXiv:2211.04728. doi:10.1038/s41567-023-02226-w.

[42] Bryan Eastin and Emanuel Knill. Restrictions on Transversal Encoded Quantum Gate Sets. *Physical Review Letters*, 102(11), March 2009. arXiv:0811.4262. doi:10.1103/physrevlett.102.110502.

[43] Michael H. Freedman and David A. Meyer. Projective Plane and Planar Quantum Codes. *Foundations of Computational Mathematics*, 1(3):325–332, July 2001. arXiv:quant-ph/9810055. doi:10.1007/s102080010013.

[44] Sergey B Bravyi and A Yu Kitaev. Quantum codes on a lattice with boundary. *arXiv preprint*, 1998. arXiv:quant-ph/9811052.

[45] H. Bombin and M. A. Martin-Delgado. Optimal resources for topological two-dimensional stabilizer codes: Comparative study. *Physical Review A*, 76(1), July 2007. arXiv:quant-ph/0703272. doi:10.1103/physreva.76.012305.

[46] Jack Edmonds. Paths, Trees, and Flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/cjm-1965-045-4.

[47] Sergey Bravyi, Martin Suchara, and Alexander Vargo. Efficient algorithms for maximum likelihood decoding in the surface code. *Physical Review A*, 90(3), September 2014. arXiv:1405.4883. doi:10.1103/physreva.90.032326.

[48] Joschka Roffe, David R. White, Simon Burton, and Earl Campbell. Decoding across the quantum low-density parity-check code landscape. *Physical Review Research*, 2(4), December 2020. arXiv:2005.07016. doi:10.1103/physrevresearch.2.043423.

[49] Nicolas Delfosse and Naomi H. Nickerson. Almost-linear time decoding algorithm for topological codes. *Quantum*, 5:595, December 2021. arXiv:1709.06218. doi:10.22331/q-2021-12-02-595.

[50] Oscar Higgott. PyMatching: A Python Package for Decoding Quantum Codes with Minimum-Weight Perfect Matching. *ACM Transactions on Quantum Computing*, 3(3):1–16, June 2022. arXiv:2105.13082. doi:10.1145/3505637.

[51] Yue Wu and Lin Zhong. Fusion Blossom: Fast MWPM Decoders for QEC. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, page 928–938. IEEE, September 2023. arXiv:2305.08307. doi:10.1109/qce57702.2023.00107.

[52] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, September 2002. arXiv:quant-ph/0110143. doi:10.1063/1.1499754.

[53] D.S. Wang, A.G. Fowler, A.M. Stephens, and L.C.L. Hollenberg. Threshold error rates for the toric and planar codes. *Quantum Information and Computation*, 10(5 & 6):456–469, May 2010. arXiv:0905.0531. doi:10.26421/qic10.5-6-6.

[54] H Bombin. Clifford gates by code deformation. *New Journal of Physics*, 13(4):043005, April 2011. arXiv:1006.5260. doi:10.1088/1367-2630/13/4/043005.

[55] Theodore J. Yoder and Isaac H. Kim. The surface code with a twist. *Quantum*, 1:2, April 2017. arXiv:1612.04795. doi:10.22331/q-2017-04-25-2.

[56] Benjamin J. Brown, Katharina Laubscher, Markus S. Kesselring, and James R. Wootton. Poking Holes and Cutting Corners to Achieve Clifford Gates with the Surface Code. *Physical Review X*, 7(2), May 2017. arXiv:1609.04673. doi:10.1103/physrevx.7.021029.

[57] Daniel Litinski and Felix von Oppen. Lattice Surgery with a Twist: Simplifying Clifford Gates of Surface Codes. *Quantum*, 2:62, May 2018. arXiv:1709.02318. doi:10.22331/q-2018-05-04-62.

[58] Robert Raussendorf and Jim Harrington. Fault-Tolerant Quantum Computation with High Threshold in Two Dimensions. *Physical Review Letters*, 98(19), May 2007. arXiv:quant-ph/0610082. doi:10.1103/physrevlett.98.190504.

[59] H Bombin and M A Martin-Delgado. Quantum measurements and gates by code deformation. *Journal of Physics A: Mathematical and Theoretical*, 42(9):095302, February 2009. arXiv:0704.2540. doi:10.1088/1751-8113/42/9/095302.

[60] Sergey Bravyi and Jeongwan Haah. Magic-state distillation with low overhead. *Physical Review A*, 86(5), November 2012. arXiv:1209.2426. doi:10.1103/physreva.86.052329.

[61] Bob Coecke and Ross Duncan. *Interacting Quantum Observables*, page 298–310. Springer Berlin Heidelberg, 2008. doi:10.1007/978-3-540-70583-3_25.

[62] S. Abramsky and B. Coecke. A categorical semantics of quantum protocols. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science*, page 415–425. IEEE, 2004. arXiv:quant-ph/0402130. doi:10.1109/lics.2004.1319636.

[63] Miriam Backens. The ZX-calculus is complete for stabilizer quantum mechanics. *New Journal of Physics*, 16(9):093021, September 2014. arXiv:1307.7025. doi:10.1088/1367-2630/16/9/093021.

[64] Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. A Complete Axiomatisation of the ZX-Calculus for Clifford+T Quantum Mechanics. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, page 559–568. ACM, July 2018. arXiv:1705.11151. doi:10.1145/3209108.3209131.

[65] Kang Feng Ng and Quanlong Wang. A universal completion of the ZX-calculus. *arXiv preprint*, 2017. arXiv:1706.09877.

[66] Ross Duncan and Simon Perdrix. *Rewriting Measurement-Based Quantum Computations with Generalised Flow*, page 285–296. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-14162-1_24.

[67] Ross Duncan, Aleks Kissinger, Simon Perdrix, and John van de Wetering. Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus. *Quantum*, 4:279, June 2020. arXiv:1902.03178. doi:10.22331/q-2020-06-04-279.

[68] Aleks Kissinger and John van de Wetering. Reducing the number of non-Clifford gates in quantum circuits. *Physical Review A*, 102(2), August 2020. doi:10.1103/physreva.102.022406.

[69] Aleks Kissinger and John van de Wetering. Simulating quantum circuits with ZX-calculus reduced stabiliser decompositions. *Quantum Science and Technology*, 7(4):044001, July 2022. arXiv:2109.01076. doi:10.1088/2058-9565/ac5d20.

[70] Tristan Cam and Simon Martiel. Speeding up quantum circuits simulation using ZX-Calculus. *arXiv preprint*, 2023. arXiv:2305.02669.

[71] Craig Gidney and Austin G. Fowler. Efficient magic state factories with a catalyzed $|CCZ\rangle$ to $2|T\rangle$ transformation. *Quantum*, 3:135, April 2019. arXiv:1812.01238. doi:10.22331/q-2019-04-30-135.

[72] Michael Hanks, Marta P. Estarellas, William J. Munro, and Kae Nemoto. Effective Compression of Quantum Braided Circuits Aided by ZX-Calculus. *Physical Review X*, 10(4), November 2020. arXiv:1912.11503. doi:10.1103/physrevx.10.041030.

[73] Hector Bombin, Daniel Litinski, Naomi Nickerson, Fernando Pastawski, and Sam Roberts. Unifying flavors of fault tolerance with the ZX calculus. *Quantum*, 8, June 2024. arXiv:2303.08829. doi:10.22331/q-2024-06-18-1379.

[74] Niel de Beaudrap and Dominic Horsman. The ZX calculus is a language for surface code lattice surgery. *Quantum*, 4:218, January 2020. arXiv:1704.08670. doi:10.22331/q-2020-01-09-218.

[75] Niel de Beaudrap, Ross Duncan, Dominic Horsman, and Simon Perdrix. Pauli Fusion: a Computational Model to Realise Quantum Transformations from ZX Terms. *Electronic Proceedings in Theoretical Computer Science*, 318:85–105, May 2020. arXiv:1904.12817. doi:10.4204/eptcs.318.6.

[76] Archimedes Pavlidis and Dimitris Gizopoulos. Fast quantum modular exponentiation architecture for Shor's factoring alogrithm. *Quantum Information and Computation*, 14(7 & 8):649–682, May 2014. arXiv:1207.0511. doi:10.26421/qic14.7-8-8.

[77] Vlatko Vedral, Adriano Barenco, and Artur Ekert. Quantum networks for elementary arithmetic operations. *Physical Review A*, 54(1):147–153, July 1996. arXiv:quant-ph/9511018. doi:10.1103/physreva.54.147.

[78] Steven A. Cuccaro, Thomas G. Draper, Samuel A. Kutin, and David Petrie Moulton. A new quantum ripple-carry addition circuit. *arXiv preprint quant-ph/0410184*, 2004. arXiv:quant-ph/0410184.

[79] Y. Takahashi and N. Kunihiro. A linear-size quantum circuit for addition with no ancillary qubits. *Quantum Information and Computation*, 5(6):440–448, September 2005. doi:10.26421/qic5.6-2.

[80] Feng Wang, Mingxing Luo, Huiran Li, Zhiguo Qu, and Xiaojun Wang. Improved quantum ripple-carry addition circuit. *Science China Information Sciences*, 59(4), February 2016. doi:10.1007/s11432-015-5411-x.

[81] Craig Gidney. Halving the cost of quantum addition. *Quantum*, 2:74, June 2018. arXiv:1709.06648. doi:10.22331/q-2018-06-18-74.

[82] T.G. Draper, S.A. Kutin, E.M. Rains, and K.M. Svore. A logarithmic-depth quantum carry-lookahead adder. *Quantum Information and Computation*, 6(4 & 5):351–369, July 2006. doi:10.26421/qic6.4-5-4.

[83] Y. Takahashi and N. Kunihiro. A fast quantum circuit for addition with few qubits. *Quantum Information and Computation*, 8(6 & 7):636–649, July 2008. doi:10.26421/qic8.6-7-5.

[84] Torben Ægidius Mogensen. *Reversible In-Place Carry-Lookahead Addition with Few Ancillae*, page 224–237. Springer International Publishing, 2019. doi:10.1007/978-3-030-21500-2_14.

[85] Thomas G Draper. Addition on a quantum computer. *arXiv preprint*, 2000. arXiv:quant-ph/0008033.

[86] J. Proos and Ch. Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *Quantum Information and Computation*, 3(4):317–344, July 2003. arXiv:quant-ph/0301141. doi:10.26421/qic3.4-3.

[87] Donny Cheung, Dmitri Maslov, Jimson Mathew, and Dhiraj K Pradhan. On the design and optimization of a quantum polynomial-time attack on elliptic curve cryptography. In *Theory of Quantum Computation, Communication, and Cryptography: Third Workshop, TQC 2008 Tokyo, Japan, January 30-February 1, 2008. Revised Selected Papers 3*, pages 96–104. Springer, 2008. arXiv:0710.1093. doi:10.1007/978-3-540-89304-2_9.

[88] P.R. Kaye. Optimized quantum implementation of elliptic curve arithmetic over binary fields. *Quantum Information and Computation*, 5(6):474–491, September 2005. arXiv:quant-ph/0407095. doi:10.26421/qic5.6-6.

[89] Brittanney Amento, Martin Rotteler, and Rainer Steinwalds. Efficient quantum circuits for binary elliptic curve arithmetic: reducing T-gate complexity. *Quantum Information and Computation*, 13(7 & 8):631–644, May 2013. arXiv:1209.6348. doi:10.26421/qic13.7-8-5.

[90] Parshuram Budhathoki and Rainer Steinwandt. Automatic synthesis of quantum circuits for point addition on ordinary binary elliptic curves. *Quantum Information Processing*, 14(1):201–216, October 2014. arXiv:1401.2437. doi:10.1007/s11128-014-0851-6.

[91] Gustavo Banegas, Daniel J. Bernstein, Iggy Van Hoof, and Tanja Lange. Concrete quantum cryptanalysis of binary elliptic curves. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, page 451–472, December 2020. Preprint available from https://eprint.iacr.org/2020/1296. doi:10.46586/tches.v2021.i1.451-472.

[92] Dedy Septono Catur Putranto, Rini Wisnu Wardhani, Harashta Tatimma Larasati, and Howon Kim. Another Concrete Quantum Cryptanalysis of Binary Elliptic Curves. Cryptology ePrint Archive, Paper 2022/501, 2022. URL: https://eprint.iacr.org/2022/501.

[93] Hyeonhak Kim and Seokhie Hong. New space-efficient quantum algorithm for binary elliptic curves using the optimized division algorithm. *Quantum Information Processing*, 22(6):237, June 2023. arXiv:2303.06570. doi:10.1007/s11128-023-03991-6.

[94] Ren Taguchi and Atsushi Takayasu. Concrete quantum cryptanalysis of binary elliptic curves via addition chain. *Quantum Information Processing*, 23(4):122, March 2024. arXiv:2303.06570. doi:10.1007/s11128-024-04323-y.

[95] Edoardo D. Mastrovito. *VLSI designs for multiplication over finite fields $GF(2^m)$*, page 297–309. Springer Berlin Heidelberg, 1989. doi:10.1007/3-540-51083-4_67.

[96] Edoardo D. Mastrovito. *VLSI Architectures for Computation in Galois Fields*. Phd thesis, Linköping University, Linköping, Sweden, 1991.

[97] Martin Rotteler and Rainer Steinwandt. A quantum circuit to find discrete logarithms on ordinary binary elliptic curves in depth $O(log^2 n)$. *Quantum Information and Computation*, 14(9 & 10):888–900, July 2014. arXiv:1306.1161. doi:10.26421/qic14.9-10-11.

[98] Anatolii Alekseevich Karatsuba and Yu P Ofman. Multiplication of many-digital numbers by automatic computers. In *Doklady Akademii Nauk*, volume 145, pages 293–294. Russian Academy of Sciences, 1962.

[99] Shane Kepley and Rainer Steinwandt. Quantum circuits for $\mathbb{F}_{2^n}$-multiplication with sub-quadratic gate count. *Quantum Information Processing*, 14(7):2373–2386, May 2015. doi:10.1007/s11128-015-0993-1.

[100] Iggy van Hoof. Space-efficient quantum multiplication polynomials for binary finite fields with sub-quadratic Toffoli gate count. *Quantum Information and Computation*, 20(9 & 10):721–735, August 2020. arXiv:1910.02849. doi:10.26421/qic20.9-10-1.

[101] Dedy Septono Catur Putranto, Rini Wisnu Wardhani, Harashta Tatimma Larasati, Janghyun Ji, and Howon Kim. Depth-Optimization of Quantum Cryptanalysis on Binary Elliptic Curves. *IEEE Access*, 11:45083–45097, 2023. doi:10.1109/access.2023.3273601.

[102] A. Reyhani-Masoleh and M.A. Hasan. Low complexity bit parallel architectures for polynomial basis multiplication over $GF(2^m)$. *IEEE Transactions on Computers*, 53(8):945–959, August 2004. doi:10.1109/tc.2004.47.

[103] Samuel Kutin, David Moulton, and Lawren Smithline. Computation at a distance. *Chicago Journal of Theoretical Computer Science*, 13(1), 2007. arXiv:quant-ph/0701194. doi:10.4086/cjtcs.2007.001.

[104] Matthew Amy, Dmitri Maslov, and Michele Mosca. Polynomial-time T-depth optimization of Clifford+ T circuits via matroid partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(10):1476–1489, October 2014. arXiv:1303.2042. doi:10.1109/tcad.2014.2341953.

[105] Nabila Abdessaied, Mathias Soeken, and Rolf Drechsler. Quantum Circuit Optimization by Hadamard Gate Reduction. In Shigeru Yamashita and Shin-ichi Minato, editors, *Reversible Computation*, pages 149–162. Springer International Publishing, 2014. doi:1007/978-3-319-08494-7_12.

[106] Matthew Amy and Michele Mosca. T-count optimization and Reed–Muller codes. *IEEE Transactions on Information Theory*, 65(8):4771–4784, August 2019. arXiv:1601.07363. doi:10.1109/tit.2019.2906374.

[107] Yunseong Nam, Neil J Ross, Yuan Su, Andrew M Childs, and Dmitri Maslov. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Information*, 4(1):1–12, May 2018. arXiv:1710.07345. doi:10.1038/s41534-018-0072-4.

[108] Luke E Heyfron and Earl T Campbell. An efficient quantum compiler that reduces T count. *Quantum Science and Technology*, 4(1):015004, September 2018. arXiv:1712.01557. doi:10.1088/2058-9565/aad604.

[109] Aleks Kissinger and John van de Wetering. Reducing the number of non-Clifford gates in quantum circuits. *Physical Review A*, 102(2), August 2020. arXiv:1903.10477. doi:10.1103/physreva.102.022406.

[110] Fang Zhang and Jianxin Chen. Optimizing T gates in Clifford+T circuit as $\pi/4$ rotations around Paulis. *arXiv preprint*, 2019. arXiv:1903.12456.

[111] Niel de Beaudrap, Xiaoning Bian, and Quanlong Wang. Techniques to Reduce $\pi/4$-Parity-Phase Circuits, Motivated by the ZX Calculus. *arXiv preprint*, 2019. arXiv:1911.09039.

[112] Niel de Beaudrap, Xiaoning Bian, and Quanlong Wang. Fast and Effective Techniques for T-Count Reduction via Spider Nest Identities. In *15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2020)*, volume 158, pages 11:1–11:23, 2020. arXiv:2004.05164. doi:10.4230/LIPIcs.TQC.2020.11.

[113] Anthony Munson, Bob Coecke, and Quanlong Wang. AND-gates in ZX-calculus: Spider Nest Identities and QBC-completeness. *Electronic Proceedings in Theoretical Computer Science*, 340:230–255, September 2021. arXiv:1910.06818. doi:10.4204/eptcs.340.12.

[114] Michele Mosca and Priyanka Mukhopadhyay. A polynomial time and space heuristic algorithm for T-count. *Quantum Science and Technology*, 7(1):015003, October 2021. arXiv:2006.12440. doi:10.1088/2058-9565/ac2d3a.

[115] Sergey Bravyi and David Gosset. Improved Classical Simulation of Quantum Circuits Dominated by Clifford Gates. *Physical Review Letters*, 116(25), June 2016. arXiv:1601.07601. doi:10.1103/physrevlett.116.250501.

[116] Sergey Bravyi, Dan Browne, Padraic Calpin, Earl Campbell, David Gosset, and Mark Howard. Simulation of quantum circuits by low-rank stabilizer decompositions. *Quantum*, 3:181, September 2019. arXiv:1808.00128. doi:10.22331/q-2019-09-02-181.

[117] Hammam Qassim, Hakop Pashayan, and David Gosset. Improved upper bounds on the stabilizer rank of magic states. *Quantum*, 5:606, December 2021. arXiv:2106.07740. doi:10.22331/q-2021-12-20-606.

[118] Aleks Kissinger and John van de Wetering. Simulating quantum circuits with ZX-calculus reduced stabiliser decompositions. *Quantum Science and Technology*, 7(4):044001, July 2022. arXiv:2109.01076. doi:10.1088/2058-9565/ac5d20.

[119] Aleks Kissinger, John van de Wetering, and Renaud Vilmart. Classical Simulation of Quantum Circuits with Partial and Graphical Stabiliser Decompositions. 232:5:1–5:13, 2022. arXiv:2202.09202. doi:10.4230/LIPIcs.TQC.2022.5.

[120] G. Seroussi and A. Lempel. Maximum likelihood decoding of certain Reed - Muller codes (Corresp.). *IEEE Transactions on Information Theory*, 29(3):448–450, May 1983. doi:10.1109/tit.1983.1056662.

[121] Michael J. Bremner, Richard Jozsa, and Dan J. Shepherd. Classical simulation of commuting quantum computations implies collapse of the polynomial hierarchy. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 467(2126):459–472, August 2010. arXiv:1005.1407. doi:10.1098/rspa.2010.0301.

[122] Matthew Amy, Parsiad Azimzadeh, and Michele Mosca. On the controlled-NOT complexity of controlled-NOT–phase circuits. *Quantum Science and Technology*, 4(1):015002, September 2018. arXiv:1712.01859. doi:10.1088/2058-9565/aad8ca.

[123] Dmitri Maslov and Martin Roetteler. Shorter Stabilizer Circuits via Bruhat Decomposition and Quantum Circuit Transformations. *IEEE Transactions on Information Theory*, 64(7):4729–4738, July 2018. arXiv:1705.09176. doi:10.1109/tit.2018.2825602.

[124] Timothée Goubault de Brugière, Simon Martiel, and Christophe Vuillot. A graph-state based synthesis framework for Clifford isometries. *arXiv preprint*, 2022. arXiv:2212.06928.

[125] Craig Gidney. Stim: a fast stabilizer circuit simulator. *Quantum*, 5:497, July 2021. arXiv:2103.02202. doi:10.22331/q-2021-07-06-497.

[126] Will Simmons. Relating Measurement Patterns to Circuits via Pauli Flow. *Electronic Proceedings in Theoretical Computer Science*, 343:50–101, September 2021. arXiv:2109.05654. doi:10.4204/eptcs.343.4.

[127] Matthew Amy. Feynman. https://github.com/meamy/feynman.

[128] Dmitri Maslov. Reversible Logic Synthesis Benchmarks page. http://webhome.cs.uvic.ca/~dmaslov. Accessed February 2023.

[129] Anubhab Baksi and Kyungbae Jang. *Quantum Implementation and Analysis of DEFAULT*, page 39–49. Springer Nature Singapore, 2024. doi:10.1007/978-981-97-0025-7_5.

[130] Xiaoning Bian. STOMP-code. https://github.com/onestruggler/stomp-code/tree/8df4f46228c2f413e0cf5f8b6d25c20b6460fc0e.

[131] Francisco JR Ruiz, Tuomas Laakkonen, Johannes Bausch, Matej Balog, Mohammadamin Barekatain, Francisco JH Heras, Alexander Novikov, Nathan Fitzpatrick, Bernardino Romera-Paredes, John van de Wetering, et al. Quantum circuit optimization with alphatensor. *arXiv preprint*, 2024. arXiv:2402.14396.

[132] Matthew Amy and Vlad Gheorghiu. staq—A full-stack quantum processing toolkit. *Quantum Science and Technology*, 5(3):034016, 2020. arXiv:1912.06070. doi:10.1088/2058-9565/ab9359.

[133] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. t|ket⟩: a retargetable compiler for NISQ devices. *Quantum Science and Technology*, 6(1):014003, 2020. arXiv:2003.10611. doi:10.1088/2058-9565/ab8e92.

[134] Simon Martiel and Timothée Goubault de Brugière. Architecture aware compilation of quantum circuits via lazy synthesis. *Quantum*, 6:729, 2022. arXiv:2012.09663. doi:10.22331/q-2022-06-07-729.

[135] Aleks Kissinger and John van de Wetering. PyZX: Large Scale Automated Diagrammatic Reasoning. volume 318, page 229–241. Open Publishing Association, May 2020. arXiv:1904.04735. doi:10.4204/eptcs.318.14.

[136] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014. arXiv:1411.4028.

[137] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5(1), July 2014. arXiv:1304.3061. doi:10.1038/ncomms5213.

[138] John van de Wetering, Richie Yeung, Tuomas Laakkonen, and Aleks Kissinger. Optimal compilation of parametrised quantum circuits. 2024. arXiv:2401.12877.

[139] Robert Raussendorf, Jim Harrington, and Kovid Goyal. Topological fault-tolerance in cluster state quantum computation. *New Journal of Physics*, 9(6):199, 2007. doi:10.1088/1367-2630/9/6/199.

[140] Austin G Fowler, Ashley M Stephens, and Peter Groszkowski. High-threshold universal quantum computation on the surface code. *Physical Review A*, 80(5):052312, 2009. arXiv:0803.0272. doi:10.1103/physreva.80.052312.

[141] Michael E Beverland, Aleksander Kubica, and Krysta M Svore. Cost of universality: A comparative study of the overhead of state distillation and code switching with color codes. *PRX Quantum*, 2(2):020341, 2021. arXiv:2101.02211. doi:10.1103/prxquantum.2.020341.

[142] Austin G. Fowler. Time-optimal quantum computation. *arXiv preprint*, 2013. arXiv:1210.4626.

[143] Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(6):818–830, 2013. arXiv:1206.0758. doi:10.1109/tcad.2013.2244643.

[144] Peter Selinger. Quantum circuits of T-depth one. *Physical Review A*, 87(4):042302, 2013. arXiv:1210.0974. doi:10.1103/PhysRevA.87.042302.

[145] Philipp Niemann, Anshu Gupta, and Rolf Drechsler. T-depth optimization for fault-tolerant quantum circuits. In *2019 IEEE 49th International Symposium on Multiple-Valued Logic (ISMVL)*, pages 108–113. IEEE, 2019. doi:10.1109/ismvl.2019.00027.

[146] Vlad Gheorghiu, Michele Mosca, and Priyanka Mukhopadhyay. A (quasi-) polynomial time heuristic algorithm for synthesizing T-depth optimal circuits. *npj Quantum Information*, 8(1):110, 2022. arXiv:2101.03142. doi:10.1038/s41534-022-00624-1.

[147] Peter Selinger. Efficient Clifford+T approximation of single-qubit operators. *Quantum Information & Computation*, 15(1-2):159–180, 2015. arXiv:1212.6253. doi:10.26421/qic15.1-2-10.

[148] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Asymptotically optimal approximation of single qubit unitaries by Clifford and T circuits using a constant number of ancillary qubits. *Physical review letters*, 110(19):190502, 2013. arXiv:1212.0822. doi:10.1103/physrevlett.110.190502.

[149] Neil J. Ross and Peter Selinger. Optimal ancilla-free Clifford+T approximation of z-rotations. *Quantum Information and Computation*, 16(11–12):901–953, sep 2016. arXiv:1403.2975. doi:10.26421/qic16.11-12-1.

[150] Alex Bocharov, Martin Roetteler, and Krysta M Svore. Efficient synthesis of probabilistic quantum circuits with fallback. *Physical Review A*, 91(5):052317, 2015. arXiv:1409.3552. doi:10.1103/physreva.91.052317.

[151] Matthew B. Hastings. Turning gate synthesis errors into incoherent errors. *Quantum Information and Computation*, 17:488–494, April 2017. arXiv:1612.01011. doi:10.26421/qic17.5-6-7.

[152] Earl Campbell. Shorter gate sequences for quantum computing by mixing unitaries. *Physical Review A*, 95(4):042306, 2017. arXiv:1612.02689. doi:10.1103/physreva.95.042306.

[153] Vadym Kliuchnikov, Kristin Lauter, Romy Minko, Adam Paetznick, and Christophe Petit. Shorter quantum circuits via single-qubit gate approximation. *Quantum*, 7:1208, December 2023. arXiv:2203.10064. doi:10.22331/q-2023-12-18-1208.

[154] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Fast and efficient exact synthesis of single-qubit unitaries generated by Clifford and T gates. *Quantum Information & Computation*, 13(7-8):607–630, 2013. arXiv:1206.5236. doi:10.26421/qic13.7-8-4.

[155] Witalis Domitrz. On the Verge to Improve Technique of T-count Reduction via Spider Nest Identities. Master's thesis, University of Oxford, 2021.

[156] Ewout Van Den Berg and Kristan Temme. Circuit optimization of Hamiltonian simulation by simultaneous diagonalization of Pauli clusters. *Quantum*, 4:322, 2020. arXiv:2003.13599. doi:10.22331/q-2020-09-12-322.

[157] Earl T Campbell and Mark Howard. Unified framework for magic state distillation and multiqubit gate synthesis with reduced resource cost. *Physical Review A*, 95(2):022316, 2017. arXiv:1606.01906. doi:10.1103/physreva.95.022316.

[158] Daniel Gottesman and Isaac L Chuang. Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations. *Nature*, 402(6760):390–393, 1999. arXiv:quant-ph/9908010. doi:10.1038/46503.

[159] Christopher M Dawson, Henry L Haselgrove, Andrew P Hines, Duncan Mortimer, Michael A Nielsen, and Tobias J Osborne. Quantum computing and polynomial equations over Z_2. *Quantum Information and Computation*, 5(2):102–112, May 2005. arXiv:quant-ph/0408129. doi:10.26421/qic5.2-2.

[160] Ashley Montanaro. Quantum circuits and low-degree polynomials over $\mathbb{F}_2$. *Journal of Physics A: Mathematical and Theoretical*, 50(8):084002, January 2017. arXiv:1607.08473. doi:10.1088/1751-8121/aa565f.

[161] Johan Håstad. Tensor rank is NP-complete. *Journal of Algorithms*, 11(4):644–654, December 1990. doi:10.1016/0196-6774(90)90014-6.

[162] John van de Wetering and Matt Amy. Optimising T-count is NP-hard. *arXiv preprint*, 2023. arXiv:2310.05958.

[163] Cody Jones. Low-overhead constructions for the fault-tolerant Toffoli gate. *Physical Review A—Atomic, Molecular, and Optical Physics*, 87(2):022328, 2013. arXiv:1212.5069. doi:10.1103/physreva.87.022328.

[164] Sergey Bravyi and Jeongwan Haah. Magic-state distillation with low overhead. *Physical Review A—Atomic, Molecular, and Optical Physics*, 86(5):052329, 2012. arXiv:1209.2426. doi:10.1103/physreva.86.052329.

[165] Guillaume Duclos-Cianci and David Poulin. Reducing the quantum-computing overhead with complex gate distillation. *Physical Review A*, 91(4):042315, 2015. arXiv:1403.5280. doi:10.1103/physreva.91.042315.

[166] Earl T Campbell and Mark Howard. Magic state parity-checker with pre-distilled components. *Quantum*, 2:56, 2018. arXiv:1709.02214. doi:10.22331/q-2018-03-14-56.

[167] Abraham Lempel. Matrix Factorization over $GF(2)$ and Trace-Orthogonal Bases of $GF(2^n)$. *SIAM Journal on Computing*, 4(2):175–186, 1975. doi:10.1137/0204014.

[168] Daniel Litinski. Magic State Distillation: Not as Costly as You Think. *Quantum*, 3:205, December 2019. arXiv:1905.06903. doi:10.22331/q-2019-12-02-205.

[169] Richard M. Karp. Reducibility Among Combinatorial Problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

[170] Yongshan Ding, Xin-Chuan Wu, Adam Holmes, Ash Wiseth, Diana Franklin, Margaret Martonosi, and Frederic T. Chong. SQUARE: Strategic Quantum Ancilla Reuse for Modular Quantum Programs via Cost-Effective Uncomputation. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, page 570–583. IEEE, May 2020. arXiv:2004.08539. doi:10.1109/isca45697.2020.00054.

[171] Fei Hua, Yuwei Jin, Yanhao Chen, Suhas Vittal, Kevin Krsulich, Lev S. Bishop, John Lapeyre, Ali Javadi-Abhari, and Eddy Z. Zhang. CaQR: A Compiler-Assisted Approach for Qubit Reuse through Dynamic Circuit. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '23, page 59–71. ACM, March 2023. doi:10.1145/3582016.3582030.

[172] Matthew DeCross, Eli Chertkov, Megan Kohagen, and Michael Foss-Feig. Qubit-Reuse Compilation with Mid-Circuit Measurement and Reset. *Physical Review X*, 13(4), December 2023. arXiv:2210.08039. URL: http://dx.doi.org/10.1103/physrevx.13.041057, doi:10.1103/physrevx.13.041057.

[173] Sebastian Brandhofer, Ilia Polian, and Kevin Krsulich. Optimal Qubit Reuse for Near-Term Quantum Computers. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, page 859–869. IEEE, September 2023. arXiv:2308.00194. doi:10.1109/qce57702.2023.00100.

[174] Movahhed Sadeghi, Soheil Khadirsharbiyani, and Mahmut Taylan Kandemir. Quantum Circuit Resizing. *arXiv preprint*, 2022. arXiv:2301.00720.

[175] Fanica Gavril. Some NP-complete problems on graphs. In *Annual Conference on Information Sciences and Systems*, pages 91–95, 1977.

[176] Nancy G. Kinnersley. The vertex separation number of a graph equals its path-width. *Information Processing Letters*, 42(6):345–350, July 1992. doi:10.1016/0020-0190(92)90234-m.

[177] Toshinobu Kashiwabara. NP-completeness of the problem of finding a minimal-cliquenumber interval graph containing a given graph as a subgraph. In *Proc. 1979 Int. Symp. Circuit Syst*, pages 657–660, 1979.

[178] T. Ohtsuki, H. Mori, E.S. Kuh, T. Kashiwabara, and T. Fujisawa. One-dimensional logic gate assignment and interval graphs. page 101106. doi:10.1109/cmpsac.1979.762474.

[179] Neil Robertson and P.D. Seymour. Graph minors. I. Excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, August 1983. doi:10.1016/0095-8956(83)90079-5.

[180] Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1–2):1–45, December 1998. doi:10.1016/s0304-3975(97)00228-4.

[181] Gabriel Bathie, Gaétan Berthe, Yoann Coudert-Osmont, David Desobry, Amadeus Reinald, and Mathis Rocton. PACE Solver Description: DreyFVS. In *17th International Symposium on Parameterized and Exact Computation*, page 28, 2022.

[182] Gérard D Cohen and Simon N Litsyn. On the covering radius of Reed-Muller codes. *Discrete Mathematics*, 106:147–155, 1992. doi:10.1016/0012-365x(92)90542-n.

Résumé

Le calcul quantique tolérant aux fautes nécessite des techniques de compilation efficaces pour traduire les algorithmes quantiques de haut niveau en opérations exécutables sur le matériel. Cette thèse présente plusieurs contributions au développement d'une pile de compilation pour le calcul quantique tolérant aux fautes.

Au niveau de la synthèse de circuits quantiques, nous introduisons de nouvelles constructions pour des opérateurs arithmétiques, notamment le premier additionneur exact sans qubit auxiliaire avec une profondeur sous-linéaire, ainsi qu'un multiplicateur pour les corps binaires offrant un faible coût espace-temps.

Pour l'optimisation de circuits quantiques, nous présentons des algorithmes pour minimiser le nombre de portes de Hadamard et de rotation dans les circuits, en prouvant leur optimalité dans certains cas. Nous proposons également des algorithmes efficaces pour réduire le nombre de portes T, une ressource critique pour le calcul quantique tolérant aux fautes. Nos méthodes permettent d'obtenir le plus petit nombre de portes T pour la plupart des circuits évalués, tout en offrant une mise à l'échelle nettement meilleure que celle des approches précédentes.

Enfin, en utilisant le ZX-calcul, nous établissons des liens entre des problèmes d'optimisation de ressources et des problèmes bien étudiés en théorie des graphes. Cela nous permet de développer de nouvelles techniques d'optimisation pour minimiser le nombre de qubits dans les modèles de calcul tolérants aux fautes, tels que la chirurgie de codes.

Mots-clés: Informatique quantique, Optimisation de circuits quantiques, Informatique quantique tolérant aux fautes, ZX-calcul.

Abstract

Large-scale fault-tolerant quantum computing requires efficient compilation techniques to translate high-level quantum algorithms into executable operations on quantum hardware. This thesis presents several contributions to the development of a compilation stack for fault-tolerant quantum computing.

At the quantum circuit synthesis level, we introduce novel constructions for some arithmetic operators, including the first ancilla-free exact quantum adder with sublinear depth and an efficient quantum multiplier for binary fields achieving state-of-the-art space-time cost.

For quantum circuit optimization, we present algorithms for minimizing the number of Hadamard and rotation gates in quantum circuits, proving their optimality in some cases. We also propose efficient algorithms for reducing the number of T gates, a critical resource in fault-tolerant quantum computing. Our methods achieve the best-known T-count reduction for most evaluated quantum circuits while offering significantly better scalability than previous approaches.

Finally, by using the ZX-calculus, we establish connections between resource optimization problems and well-studied problems in graph theory. This enables us to develop new optimization techniques for minimizing the number of qubits in fault-tolerant computational models, such as surface code lattice surgery.

Keywords: Quantum computing, Quantum circuit optimization, Fault-tolerant quantum computing, ZX-calculus.