Généralisation des techniques d'énumération pour les méthodes variationnelles quantiques

Rapport de stage de Master 2

Master de Statistique et Sciences des Données

Université de Montpellier
Faculté des Sciences (FdS)

Année universitaire 2024-2025

Auteur : Lorenzo Gaggini
Encadrants professionnels : Eric Boureau
Tuteur FdS : Elodie Brunel
Jury : Benoîte de Saporta, Nicolas Meyer, Elodie Brunel
Organisme d'accueil : LIRMM - Laboratoire d'informatique, de robotique et de microélectronique de Montpellier

Remerciements

Je tiens à exprimer ma profonde gratitude à l'ensemble des personnes qui ont contribué au bon déroulement de ce stage et à la réalisation de ce rapport.

Je remercie tout particulièrement Eric Boureau, mon encadrant professionnel au LIRMM, pour son accompagnement, ses conseils avisés et la confiance qu'il m'a accordée tout au long de ce travail. Sa disponibilité et son expertise ont été déterminantes dans l'avancée de mes travaux.

Je souhaite également exprimer ma reconnaissance à Imran Meghazi, doctorant au sein de l'équipe MAORE, pour ses échanges scientifiques, son soutien et ses nombreux conseils qui ont enrichi ce travail.

Je tiens à remercier Elodie Brunel, ma tutrice à la Faculté des Sciences, de m'avoir permis de réaliser ce stage, malgré son caractère atypique par rapport aux stages traditionnels en statistiques.

Je remercie l'ensemble des membres de l'équipe MAORE pour leur accueil chaleureux, leurs échanges stimulants et leur soutien au quotidien.

Je remercie par ailleurs Arnaud Peler et Gaylord Moreau pour leurs conseils avisés lors de leur relecture attentive de mon rapport, ainsi que pour leur soutien et l'intérêt qu'ils ont porté à mes travaux.

Enfin, j'adresse mes remerciements à l'Université de Montpellier et au LIRMM, pour m'avoir offert un environnement de travail propice à l'apprentissage et à l'épanouissement scientifique.

Résumé

Ce travail présente une généralisation des techniques d'énumération pour l'optimisation combinatoire en informatique quantique, en prenant pour cadre d'étude le Problème du Voyageur de Commerce (TSP). L'objectif est de définir une bijection explicite entre l'espace des solutions et leur représentation binaire sur circuit quantique, en optimisant l'encodage pour atteindre la borne minimale théorique $\lceil \log_2(n!) \rceil$ en nombre de qubits.

L'approche proposée est adaptée aux architectures quantiques NISQ, en tenant compte de leurs contraintes physiques et algorithmiques. Elle vise à améliorer la robustesse et la scalabilité des algorithmes variationnels, grâce à des schémas d'encodage et de décodage adaptés à la structure combinatoire du problème.

Des simulations numériques ont été réalisées afin de valider la proposition théorique, avec deux protocoles d'optimisation : F-VQE (optimisation locale par descente de gradient), et COBYLA (optimisation locale sans gradient)

Les résultats confirment la robustesse des schémas d'encodage proposés et ouvrent des perspectives pour l'optimisation quantique d'autres problèmes combinatoires, tout en clarifiant les enjeux d'énumération et d'encodage dans les algorithmes hybrides quantique-classique.

Abstract

This work presents a generalization of enumeration techniques for combinatorial optimization in quantum computing, using the Traveling Salesman Problem (TSP) as a study framework. The objective is to define an explicit bijection between the solution space and its binary representation on a quantum circuit, optimizing the encoding to reach the theoretical minimal bound $\lceil \log_2(n!) \rceil$ in terms of qubit number.

The proposed approach is adapted to NISQ quantum architectures, taking into account their physical and algorithmic constraints. It aims to improve the robustness and scalability of variational algorithms, through encoding and decoding schemes tailored to the combinatorial structure of the problem.

Numerical simulations were performed to validate the theoretical proposal, using three optimization protocols: F-VQE (local optimization via gradient descent) and COBYLA (local optimization without gradient)

The results confirm the robustness of the proposed encoding schemes and open perspectives for quantum optimization of other combinatorial problems, while clarifying the challenges of enumeration and encoding in hybrid quantum-classical algorithms.

Table des matières

Liste des Figures

Liste des Tables

Introduction

En informatique quantique, le terme **NISQ** (pour *Noisy Intermediate-Scale Quantum*) désigne une génération d'ordinateurs quantiques caractérisée par un nombre intermédiaire de qubits (typiquement entre 50 et quelques centaines), mais dont la qualité reste limitée par le bruit et les erreurs inhérentes aux opérations physiques. Contrairement aux ordinateurs quantiques idéaux, les machines NISQ ne disposent pas de correction d'erreur quantique à grande échelle : chaque opération (porte quantique, mesure, etc.) est sujette à des imperfections, et la profondeur des circuits réalisables est fortement contrainte par l'accumulation des erreurs.

L'importance du cadre NISQ réside dans le fait qu'il correspond à l'état actuel de la technologie quantique : la plupart des plateformes expérimentales disponibles aujourd'hui (circuits supraconducteurs, ions piégés, photons, etc.) sont de type NISQ. Cela implique que les algorithmes développés doivent être adaptés à ces contraintes : ils doivent minimiser le nombre de qubits utilisés, réduire la profondeur des circuits, et être robustes face au bruit. **Les méthodes variationnelles quantiques**, qui combinent optimisation classique et circuits quantiques courts, sont particulièrement pertinentes dans ce contexte, car elles exploitent au mieux les capacités limitées des machines NISQ tout en restant applicables à des problèmes réels d'optimisation combinatoire.

Cadre et Organisme d'accueil

Ce rapport présente le travail réalisé au sein du **LIRMM** (Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier), et plus particulièrement dans l'équipe **MAORE**. Cette équipe est spécialisée dans la recherche opérationnelle et l'optimisation, Deux membres de l'équipe, **Eric Bourreau** (Enseignant chercheur, responsable adjoint de l'équipe MAORE) et **Imran Meghazi** (Doctorant de l'équipe MAORE), travaillent notamment sur le développement d'algorithmes quantiques pour des problèmes combinatoires.

Le contexte de mon stage s'inscrit dans la continuité de leurs travaux : M.Bourreau et M.Meghazi ont conçu un algorithme d'optimisation variationnelle quantique (VQA) pour le problème du voyageur de commerce (TSP), encodé de manière très compacte en $log_2(n!)$ qubits. Cette approche est particulièrement novatrice et très adaptée au contexte NISQ. Toutefois, des questions subsistaient quant à l'amélioration et à la compréhension profonde de cet algorithme.

Ma mission principale a donc été de m'approprier la méthode existante, d'en étudier les fondements mathématiques et algorithmiques, de m'enrichir des savoir-faire en recherche opérationnelle et en informatique quantique, et de proposer des pistes d'amélioration ou de généralisation, que ce soit pour le TSP ou pour d'autres problèmes.

Encadrement et travail au sein de l'équipe

Au quotidien, le travail au sein de l'équipe s'est caractérisé par une collaboration étroite, un échange constant de points de vue multidisciplinaires, ainsi qu'une **grande autonomie et liberté dans la conduite de la recherche.**

Cette autonomie s'est traduite par la possibilité d'explorer librement différentes pistes théoriques et algorithmiques, d'expérimenter des approches originales, et de confronter quotidiennement mes idées aux retours et aux questionnements du groupe. La confrontation régulière des avancées et des difficultés rencontrées a favorisé une dynamique de remise en question et d'ajustement continu, permettant d'affiner les méthodes et d'enrichir la réflexion collective.

Les réunions quotidienne et les discussions informelles ont favorisé un va-et-vient permanent entre les différentes approches : chaque avancée algorithmique ou théorique était systématiquement confrontée aux contraintes pratiques du calcul quantique, aux exigences de l'optimisation combinatoire, et aux perspectives offertes par la modélisation mathématique.

Ce dialogue interdisciplinaire a été particulièrement précieux pour identifier les points de blocage, proposer des solutions innovantes, et adapter les méthodes aux réalités du terrain. **Ce mode de fonctionnement collaboratif, fondé sur l'ouverture et la complémentarité des compétences, a été un élément clé dans la formulation des résultats présentés dans ce rapport.**

Déroulé du stage

Le point de départ de mon travail a été de comprendre la méthode initialement développée. Notamment, l'exploration des algorithmes d'énumération sur lesquels elle repose, tels que le code de Gray pour les mots binaires et l'algorithme de Trotter-Johnson pour les permutations.

Au fil de mes lectures, notamment dans une revue générale sur les **codes de Gray combinatoires** [1], j'ai découvert que ces algorithmes reposent sur des structures communes, en particulier à travers l'étude des langages formels en informatique théorique, un domaine que je ne maîtrisais pas initialement.

En parallèle, j'ai également constaté dans la littérature [2] que ces codes d'énumération peuvent être interprétés comme **des parcours hamiltoniens sur des graphes de Cayley associés à des groupes finis** : l'hypercube pour le code de Gray (groupe abélien $\mathbb{Z}_2^n$) et le graphe de Cayley du groupe symétrique S_n pour Trotter-Johnson. Cette découverte m'a permis de comprendre le lien entre ces procédures via une structure algébrique commune, celle des actions de groupe sur des ensembles finis, un concept avec lequel j'étais déjà plus familier.

Cette perspective m'a amené à explorer la théorie des graphes de Cayley et la théorie des représentations [3], afin d'adapter ces outils au service de l'optimisation quantique.

En utilisant des représentations matricielles pour décrire explicitement les actions des générateurs sur l'espace des solutions, il est possible de relier les algorithmes de Gray et de Trotter par une même séquence de multiplications matricielles, en choisissant des représentations de groupe appropriées. Ce point de vue m'a permis de retrouver certains résultats de [1], comme par exemple le nombre maximal de codes de Gray en dimension 3.

Sur cette base, j'ai cherché à formaliser des correspondances entre différents algorithmes d'énumération, en combinant les codes de Gray combinatoires et la théorie des représentations des groupes finis, dans le but d'aligner au mieux les structures combinatoires avec leur encodage sur circuit quantique, et d'en exploiter la richesse pour l'optimisation.

L'aspect statistique a été central, car la conception des algorithmes variationnels quantiques relève essentiellement d'un problème d'optimisation boîte noire : il s'agit d'ajuster les paramètres d'un circuit quantique sans connaissance explicite de la structure interne de la fonction objectif, en s'appuyant sur des méthodes statistiques pour guider l'exploration et l'amélioration des performances.

Enfin, la programmation a été également très présente : il a fallu concevoir, tester et optimiser des algorithmes en parallèle de la réflexion théorique.

Résultats et Organisation du Rapport

Les travaux réalisés au cours de ce stage ont permis d'obtenir des avancées significatives, tant sur le plan pratique que théorique.

D'un point de vue pratique, l'optimisation des schémas d'encodage et la conception de nouvelles procédures d'énumération ont conduit à **une amélioration notable des performances de l'algorithme variationnel quantique initial pour le TSP**. Les simulations numériques, menées avec différents protocoles d'optimisation (F-VQE, COBYLA), ont confirmé l'efficacité des méthodes proposées, validant ainsi leur pertinence pour les architectures quantiques NISQ.

Au-delà de l'amélioration algorithmique, la contribution principale de ce travail réside dans le développement d'un formalisme original : **les représentations opératoires**.

Ce cadre théorique, inspiré de la théorie des groupes et des représentations, propose une modélisation unifiée des algorithmes d'énumération sous la forme de procédures opératoires : des suites ordonnées d'applications de générateurs sur un espace d'états, permettant de décrire explicitement la dynamique de parcours des solutions.

Ce formalisme introduit la notion d'isomorphisme de représentations opératoires, qui permet d'établir des correspondances structurelles entre des algorithmes issus de domaines différents (par exemple, entre le code de Gray sur l'hypercube et l'algorithme de Trotter-Johnson sur le

groupe symétrique), et de concevoir des encodages binaires optimaux pour les circuits quantiques.

La section 2.1 présente l'un des premiers résultats essentiels issus du formalisme proposé : la construction explicite d'isomorphismes partiels entre représentations opératoires périodiques. Plus précisément, on établit une correspondance directe entre les cycles hamiltoniens du groupe symétrique S_4 et ceux de l'hypercube C_3, reliant ainsi l'algorithme de Trotter-Johnson pour les permutations et le code de Gray pour les mots binaires. Cette relation est illustrée dans le tableau Table 1 et visualisée en Figure 5.

Au-delà des résultats obtenus sur des cas particuliers, le formalisme des représentations opératoires développé au cours de ce stage ouvre la voie à des généralisations d'intérêt. En particulier, la méthode de partition de l'espace des labels binaires en blocs, présentée dans la partie 2.2, permet d'établir un parallèle naturel avec la décomposition de l'espace des solutions en classes d'orbites sous l'action de sous-groupes du groupe symétrique S_n.

Cette approche, présentée dans la partie 2.3, consiste à associer à des blocs de l'encodage binaire une classe d'équivalence de solutions sous l'action d'un groupe. Grâce à cette construction, il devient possible de définir des isomorphismes explicites entre la dynamique de parcours des labels binaires et celle des solutions du problème.

Cette correspondance structurelle permet non seulement d'optimiser l'encodage pour les circuits quantiques, mais aussi de transférer des techniques d'énumération et d'optimisation entre des contextes très différents, tout en préservant les propriétés essentielles des algorithmes sous-jacents.

Ce formalisme se distingue par sa dimension appliquée et multidisciplinaire : il vise avant tout à faciliter le transfert de méthodes et le design d'algorithmes quantiques, en mobilisant des outils issus des mathématiques, de l'informatique et de la physique. **L'objectif n'est pas de proposer une avancée théorique fondamentale, mais de permettre une appropriation rapide et rigoureuse des concepts existants dans les différentes littératures, afin de les adapter aux besoins concrets de l'optimisation quantique.**

Les résultats présentés en partie 3 correspondent **aux versions les plus abouties identifiées au cours de ce travail.** Toutefois, de nombreuses alternatives méthodologiques et expérimentations complémentaires ont également été menées. Un lien vers un dépôt Git est fourni en annexe, permettant d'accéder à l'ensemble des expériences additionnelles qui, bien qu'elles ne relèvent pas directement du cadre principal du rapport, présentent un intérêt scientifique notable et offrent des perspectives enrichissantes pour la poursuite des recherches dans ce domaine.

Partie 1 : Modélisation d'un problème d'optimisation en informatique quantique

---

Afin de mieux respecter les contraintes de longueur du rapport, le détail de la partie quantique a été placé en annexe. On peut s'y référer pour mieux comprendre d'où viennent les limitations NISQ. Ce n'est pas nécessaire pour la lecture principale, mais cela peut être utile si l'on souhaite approfondir la compréhension des enjeux techniques.

L'optimisation combinatoire est un domaine où l'informatique quantique pourrait apporter des progrès majeurs. Divers paradigmes et algorithmes ont été développés pour exploiter les capacités des systèmes quantiques, notamment dans le contexte des architectures actuelles NISQ.

On distingue principalement deux grandes familles d'approches :

- **Modèles purement quantiques** : Certains algorithmes, comme la recherche de Grover[4], s'exécutent entièrement sur ordinateur quantique, sans intervention classique pendant le calcul. L'état quantique est préparé en superposition sur toutes les solutions, puis des opérations unitaires amplifient la probabilité de la solution recherchée, qui est lue à la mesure finale. Ce modèle exploite directement la vitesse du calcul quantique, mais il reste difficile à mettre en oeuvre et demande beaucoup de ressources, ce qui le limite sur les machines actuelles.
- **Modèles hybrides quantique-classique** : D'autres algorithmes combinent les ressources quantiques et classiques pour l'optimisation. Typiquement, le circuit quantique est paramétré par un vecteur de paramètres réels $\theta = (\theta_1, \dots, \theta_p)$, qui contrôlent les portes du circuit agissant sur les qubits (par exemple, des degrés de rotation, des angles de phase, des inversions). Le processus d'optimisation s'effectue de façon itérative : à chaque étape, un algorithme maître sur la machine classique (comme une descente de gradient, des algorithmes génétiques, ou des méthodes bayésiennes, par exemple) propose une nouvelle valeur des paramètres θ. Le circuit quantique, avec ces paramètres, est alors exécuté sur la machine quantique ; on mesure l'état final en sortie du circuit et les résultats de la mesure sont renvoyés à l'algorithme maître sur la machine classique, qui utilise ces informations pour mettre à jour les paramètres selon une règle d'optimisation (calcul du gradient, heuristique, etc.). Ce cycle est répété jusqu'à ce qu'un critère d'arrêt soit atteint (nombre d'itérations, convergence de la fonction coût, etc.). Ce schéma hybride permet de tirer parti de la puissance du calcul quantique pour explorer efficacement l'espace des solutions, tout en bénéficiant de la

flexibilité et de la robustesse des méthodes d'optimisation sur machine classique pour ajuster les paramètres et contrôler la convergence. Parmi les exemples les plus connus de ce type d'approche, on trouve **les algorithmes variationnels quantiques**, qui sont particulièrement adaptés aux architectures NISQ.

On considère ici qu'un problème d'optimisation combinatoire consiste à explorer un espace discret de solutions admissibles afin d'identifier celle qui minimise une fonction de coût.

- Soit S l'ensemble des solutions possibles.
- On suppose qu'il existe une bijection entre S et un ensemble de labels $L \subset \{0,1\}^{n_s}$, où n_s est la dimension minimale permettant d'encoder S sous forme binaire.
- Le circuit quantique est défini comme une fonction $C_S : \mathbb{R}^p \to \mathcal{D}(\{0,1\}^{n_s})$, où $C_S(\theta)$ associe au vecteur de paramètres θ une distribution de probabilité sur les mots binaires de dimension n_s, estimée via échantillonnage répété de la mesure de l'état quantique en sortie du circuit.

La fonction de coût $f : S \to \mathbb{R}$ attribue une valeur quantitative à chaque solution, représentant son coût ou sa performance selon les critères du problème.

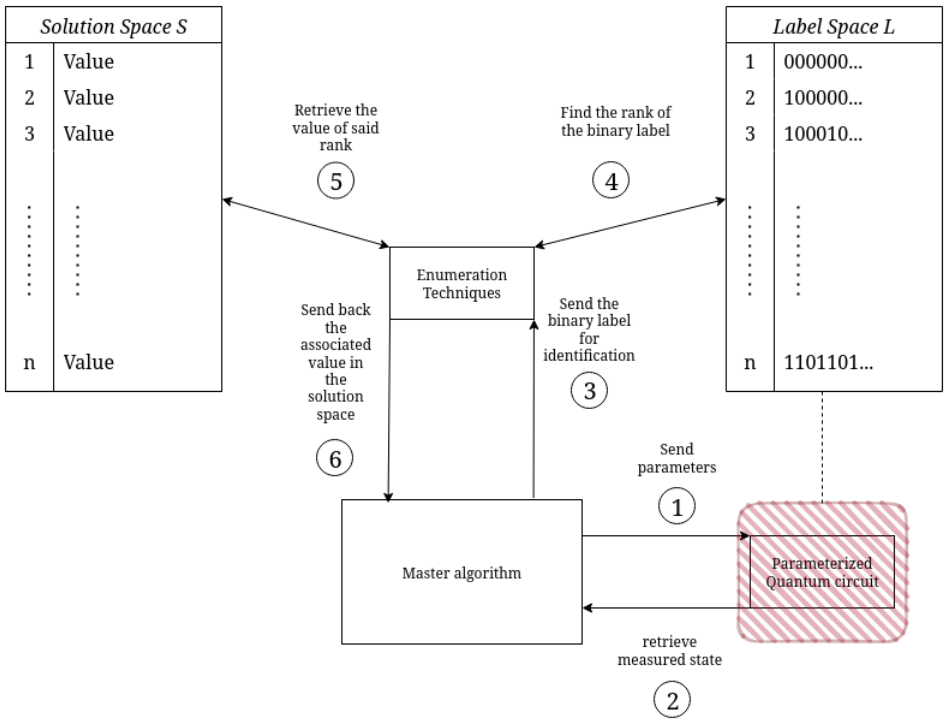

Figure 1: Schéma du protocole algorithmique. En rouge, la partie quantique ; le reste du protocole est réalisé sur ordinateur classique.

La Figure 1 synthétise le cadre d'interaction entre les ressources quantiques et classiques, tel que formalisé dans ce travail. Ce schéma sera référencé à de nombreuses reprises dans la suite pour préciser le contexte et l'étape considérée dans le processus global d'optimisation.

Dans la stratégie de l'algorithme maître M, il existe ainsi un va-et-vient constant entre les labels L produits par le circuit quantique et les solutions S qu'ils représentent : M doit sans cesse passer des labels binaires issus de la mesure quantique aux solutions concrètes du problème, puis évaluer la fonction de coût f sur ces solutions. La bijection entre L et S devient alors le pivot central du dialogue entre le calcul quantique (qui agit sur les labels) et le calcul classique (qui agit sur la fonction de coût), permettant à M d'ajuster les paramètres θ pour orienter la distribution des labels générés par le circuit en fonction des performances observées sur les solutions correspondantes.

Algorithmiquement, cette bijection est approchée par un ensemble de méthodes, que l'on désigne ici sous le nom de techniques d'énumération. L'objet principal des travaux réalisés est précisément la recherche d'une telle bijection. Il s'agit de concevoir une procédure d'encodage qui minimise le nombre de qubits, le nombre de portes quantiques, et la complexité du calcul classique, tout en permettant à l'algorithme maître M d'atteindre efficacement son critère d'arrêt.

Partie 2 : Proposition d'un cadre théorique pour les techniques d'énumération pour le TSP

Dans cette partie, on expose la méthodologie et les résultats théoriques développés durant le stage.

Le Problème du Voyageur de Commerce (TSP, pour "Travelling Salesman Problem") est un problème classique d'optimisation combinatoire. Il consiste, étant donné un ensemble de n villes et une matrice de coûts $C_{i,j}$ représentant la distance ou le coût de déplacement entre chaque paire de villes, à déterminer le chemin le moins coûteux qui permet de visiter chaque ville exactement une fois et de revenir à la ville de départ.

Pour illustrer ces concepts, on commence par introduire l'exemple suivant :

On considère un TSP à 4 villes, dont la matrice de coût est :

$$\begin{bmatrix} 0 & 10 & 15 & 20 \\ 10 & 0 & 35 & 25 \\ 15 & 35 & 0 & 30 \\ 20 & 25 & 30 & 0 \end{bmatrix}$$

Alors par exemple pour le chemin $(1, 2, 3, 4)$: on commence à la ville 1, puis on va à la ville 2 (coût $C_{1,2} = 10$), on va ensuite à la ville 3 (coût $C_{2,3} = 35$), etc.

La somme totale pour ce chemin est donc :

$$f((1,2,3,4)) = C_{1,2} + C_{2,3} + C_{3,4} + C_{4,1} = 10 + 35 + 30 + 20 = 95$$

Il existe $4! = 24$ chemins distincts, correspondant à toutes les permutations des 4 villes. L'espace des solutions S est donc l'ensemble des 24 tuples (i_1, i_2, i_3, i_4), où chaque i_k est une ville distincte de $\{1, 2, 3, 4\}$.

La dimension binaire minimale n_S pour encoder S est $\lceil \log_2(24) \rceil = 5$, soit 5 bits d'information.

La fonction de coût f associée à un chemin $s = (i_1, i_2, i_3, i_4)$ est donnée par :

$$f(s) = C_{i_1,i_2} + C_{i_2,i_3} + C_{i_3,i_4} + C_{i_4,i_1} \tag{1}$$

où $C_{i,j}$ est l'entrée correspondante dans la matrice de coût.

Il reste alors à proposer une procédure pour construire une bijection entre S et un ensemble $L \subset \{0,1\}^5$;

On rappelle que toute procédure $\mathcal{P}$ doit être à la fois théoriquement valide et réalisable en pratique.

2.0.1 Exemple introductif de procédure : la pire approche possible

Procédure $\mathcal{P}_{\text{worst}}$

1. Générer et stocker tous les $s \in S$ dans une liste ordonnée aléatoirement.
2. Générer et stocker 24 labels binaires distincts $l \in L$, choisis et ordonnée aléatoirement.
3. Associer le rang de chaque élément de S (c'est-à-dire l'ordre dans la séquence) au rang de son label dans la liste des labels.

Voici comment se déroule typiquement la procédure $\mathcal{P}_{\text{worst}}$:

- Après l'évaluation du circuit C_S, l'algorithme maître M demande le tuple associé au label binaire 01000. (Étape 2 et 3 sur la Figure 1)
- $\mathcal{P}_{\text{worst}}$ parcourt la liste L et repère que 01000 correspond au 15ème élément de L. (Étape 4 sur la Figure 1)
- $\mathcal{P}_{\text{worst}}$ sélectionne alors le 15ème élément de S et renvoie le chemin associé $(4, 1, 2, 3)$. (Étape 5 et 6 sur la Figure 1)

On pourra remarquer que cette procédure est non valide et inapplicable en pratique, mais elle permet de mettre en lumière plusieurs points essentiels pour la compréhension :

1. **Non prise en compte de la nature du circuit C_S :** Le circuit contient nécessairement aux moins 5 qubits, c'est-à-dire que, sans précision supplémentaire, 32 labels peuvent être réclamés à priori. Il existe alors 8 labels qui peuvent être réclamés par M, mais qui ne sont pas présents dans l'espace des labels L de $\mathcal{P}_{\text{worst}}$.

   Ce point est crucial et fait l'objet de la partie 2.2 : on montrera qu'il existe des moyens d'une part, de renforcer la procédure pour qu'elle soit robuste aux labels non représentatifs de S, et d'autre part, des designs particuliers de C_S qui peuvent être exploités par la procédure elle-même (c'est-à-dire indépendamment du traitement des appels au circuit que fait M) pour garantir la validité des appels de M à la procédure.

2. **Labellisations et attribution aléatoire :** Lorsque les labels sont générés aléatoirement, et que leur attribution aux solutions est réalisée de façon aléatoire, l'association entre chaque coût $f(s)$ et son label l devient elle aussi aléatoire. Dans ce contexte, il est impossible pour un algorithme maître M d'exploiter une quelconque structure : On se retrouve en fait dans un cas de **complexité black-box maximale** [5], où M n'a accès à aucune structure locale (la proximité entre labels est aléatoire) ni globale (l'ordre des solutions dans S est aléatoire et toute fonction de labellisation est choisie aléatoirement, sans propriété particulière). Dans ce cadre, on peut démontrer (voir théorème 1 de [5]) que tout algorithme maître M muni d'une telle procédure $\mathcal{P}$ ne peut pas faire mieux qu'une recherche exhaustive (c'est-à-dire faire en moyenne $\frac{\text{card}(S)+1}{2}$ appels à la procédure).

   C'est précisément l'objet de la partie 2.1 : on y propose un cadre théorique permettant définir une notion de proximité entre les solutions et leurs labels, et on montre comment ce cadre permet à tout algorithme maître de disposer d'une notion de distance pertinente sur l'espace des labels et des solutions.

3. **Inutilisabilité dans le cas général :** Lorsque S et l'espace des labels sont de dimension $n!$, cela induit un coût en mémoire et en temps prohibitif. L'un des principaux enjeux du point de vue algorithmique est de concevoir une procédure capable d'identifier le label binaire et de fournir le tuple correspondant **sans générer ni stocker** la liste complète. Typiquement, on recherche des procédures permettant de parcourir la liste sans la générer explicitement ; on parle alors de faire un **ranking/unranking**.

   Les aspects théoriques liés à l'implémentation des procédures proposées seront abordés en détail dans la partie 2.3. On y montrera notamment comment le cadre théorique élaboré permet d'évaluer, par des expérimentations, l'impact des choix d'implémentation sur la qualité des résultats.

2.1 : Sur la création d'un Isomorphisme d'énumération.

Dans cette section, on expose deux méthodes classiques d'énumération : l'algorithme de Trotter-Johnson pour les permutations et le code de Gray pour les mots binaires. Chacune

repose sur une structure combinatoire spécifique et permet, dans les deux cas, la réalisation de procédures de ranking et d'unranking en temps polynomial. Cette propriété est fondamentale pour le traitement efficace des problèmes d'optimisation combinatoire de grande dimension.

Comme illustré sur la Figure 1, l'algorithme maître doit être en mesure de décoder efficacement un label binaire obtenu après une mesure quantique, puis d'en retrouver le représentant dans l'ensemble des tuples solutions. Cette étape est cruciale pour pouvoir évaluer correctement la fonction de coût associée à ce représentant.

On s'intéresse principalement ici au ranking pour les labels (c'est-à-dire déterminer le rang d'un mot binaire dans un ordre donné), étape 4 sur la Figure 1 et l'unranking pour les permutations (retrouver la permutation correspondant à un rang donné), étape 5 sur la Figure 1.

2.1.1 Énumération des permutations : l'algorithme de Trotter-Johnson

Dans le contexte du TSP, il s'agit d'énumérer les $n!$ permutations distinctes des n villes, chaque permutation correspondant à un chemin admissible. L'algorithme de Trotter-Johnson [6] est une méthode emblématique qui génère toutes les permutations de façon ordonnée, en garantissant que chaque permutation diffère de la précédente par une transposition adjacente.

Principe et pseudocode

Soit S_n l'ensemble des permutations sur n éléments. L'algorithme construit une séquence $(\sigma_1, \sigma_2, \dots, \sigma_{n!})$ telle que pour tout k, σ_{k+1} s'obtient par une transposition adjacente appliquée à σ_k.

```
fonction TrotterJohnson(n):
    si n == 1:
        retourner [[1]]
    sinon:
        résultat ← []
        permutationsPrécédentes ← TrotterJohnson(n-1)
        pour i, p dans permutationsPrécédentes:
            si i est pair:
                pour k de 0 à longueur(p):
                    NouvellePermutation ← insérer n à la position k dans p
                    ajouter NouvellePermutation à résultat
            sinon:
                pour k de longueur(p) à 0 (décroissant):
                    NouvellePermutation ← insérer n à la position k dans p
                    ajouter NouvellePermutation à résultat
        retourner résultat
```

Exemple pour $n = 3$

L'algorithme de Trotter-Johnson génère les permutations dans l'ordre suivant, où chaque permutation diffère de la précédente par une transposition adjacente :

$$(3, 2, 1)$$
$$(2, 3, 1)$$
$$(2, 1, 3)$$
$$(1, 2, 3)$$
$$(1, 3, 2)$$
$$(3, 1, 2)$$

La technique d'enumeration associée

La technique d'énumération associée repose sur la structure de l'algorithme Johnson–Trotter. Pour le sens unranking (retrouver la permutation de rang r), la procédure est la suivante :

```
1. Initialiser un tableau `P` de taille `n` rempli de zéro.
2. Pour chaque `i` allant de `n` à `1` :
    - Calculer `q = r // i` (division entière), `p = r % i`.
    - Stocker `p` comme position d'insertion, et le sens d'insertion :
        gauche à droite si `q` est pair, droite à gauche si `q` est impair.
    - Mettre à jour `r = q`.
3. Pour chaque `i` allant de `n-1` à `0` :
    - Si le sens est gauche à droite : partir de la fin (`n-1`),
        avancer vers le début (`-1`).
    - Si le sens est droite à gauche : partir du début (`0`),
        avancer vers la fin (`+1`).
    - Avancer dans `P` en sautant les cases déjà remplies,
        jusqu'à atteindre la `p`-ième case libre.
    - Placer la valeur `i` à cette position dans `P`.
4. À la fin, le tableau `P` contient la permutation recherchée.
```

Structure hamiltonienne

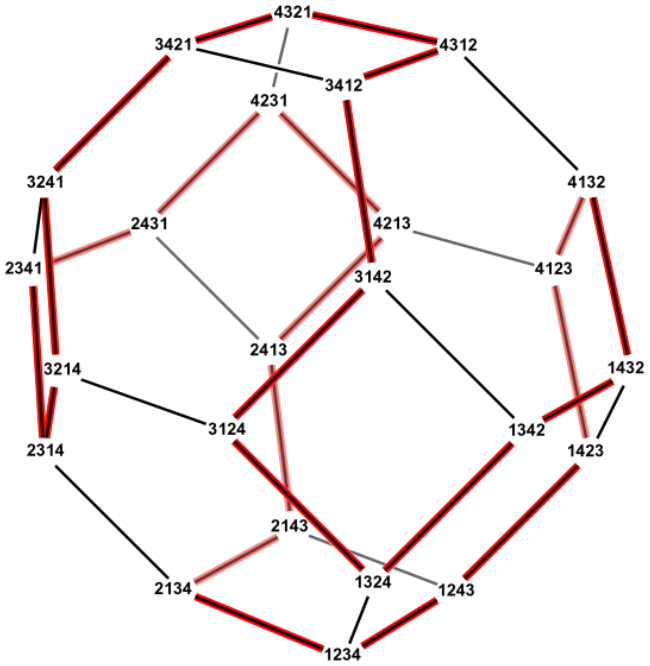

Figure 2: Cycle hamiltonien sur le graphe de Cayley du groupe S_4 avec les générateurs de transpositions adjacentes. [7]

Définition 1: Soit G un groupe fini et Σ un ensemble de générateurs de G. Le graphe de Cayley $\mathrm{Cay}(G, \Sigma)$ est le graphe dont :

- les sommets sont les éléments de G ;
- il existe une arête entre g et h si et seulement si $h = \sigma \cdot g$ pour un certain $\sigma \in \Sigma$.

Dans le cas du TSP, $G = S_n$ (le groupe des permutations de n éléments), et Σ est l'ensemble des transpositions adjacentes.

Définition 2: Un cycle hamiltonien dans un graphe est une suite de sommets telle que chaque sommet est visité exactement une fois, et que le cycle revient au sommet de départ.

L'algorithme de Trotter-Johnson génère toutes les permutations de n éléments en parcourant le graphe de Cayley de S_n selon un cycle hamiltonien, où chaque permutation consécutive diffère de la précédente par une transposition adjacente. La dernière permutation du cycle est reliée à la première, assurant ainsi un parcours exhaustif et ordonné de toutes les solutions du problème. La Figure 2 illustre ce phénomène pour $n = 4$.

2.1.2 Énumération des mots binaires : les codes de Gray

Un **code de Gray** est une séquence d'énumération des mots binaires de longueur n telle que **deux mots consécutifs ne diffèrent que par un seul bit**.

Principe et pseudocode

Il existe plusieurs façons de construire un code de Gray ; une méthode simple et élégante consiste à le définir de manière récursive. À chaque étape, on construit la séquence de Gray de dimension n à partir de celle de dimension $n-1$, en préfixant les mots existants par 0 puis en ajoutant leur miroir préfixé par 1. Cette approche garantit que deux mots consécutifs ne diffèrent que par un seul bit, et permet de généraliser facilement la construction à toute longueur de mot binaire.

```
fonction CodeGray(n):
    si n == 1:
        retourner [0, 1]
    sinon:
        séquence ← CodeGray(n-1)
        préfixe0 ← [ "0" + x for x in séquence ]
        préfixe1 ← [ "1" + x for x in séquence[::-1] ]
        retourner préfixe0 + préfixe1
```

Exemple pour $n = 3$:

- Séquence pour $n = 2$: $[00, 01, 11, 10]$
- Préfixe 0 : $[000, 001, 011, 010]$
- Préfixe 1 sur séquence inversée : $[110, 111, 101, 100]$

Séquence complète :

$$[000, 001, 011, 010, 110, 111, 101, 100]$$

La technique d'enumeration associé

La structure d'un code de Gray permet également de retrouver le mot de rang donné, ou le rang d'un mot, en temps polynomial, grâce à des algorithmes dédiés. Pour le sens ranking (retrouver le rang d'un mot donné dans la séquence de Gray), d'un mot binaire $b = b_{n-1}b_{n-2}...b_0$ dans la séquence de Gray (c'est-à-dire l'ordre d'apparition de ce mot dans le parcours du code de Gray), on utilise une conversion inverse fondée sur l'opération XOR. L'algorithme s'effectue comme suit :

```
fonction GrayRank(b):
    # b est une liste de bits [b_{n-1}, ..., b_0]
    n ← longueur(b)
    g ← liste de n zéros
    g[0] ← b[0]
    pour i de 1 à n-1:
        g[i] ← g[i-1] XOR b[i]
    # Interpréter g comme un entier décimal
    rang ← 0
    pour i de 0 à n-1:
        rang ← rang * 2 + g[i]
    retourner rang
```

Cette procédure, entièrement **linéaire** en la longueur du mot binaire, permet d'éviter la génération complète de la séquence de Gray et constitue ainsi un outil précieux pour naviguer efficacement dans l'espace des solutions.

Exemple pour $n = 3$:

On fait le ranking du mot binaire `110` de longueur 3. Pour retrouver son **rang** dans la séquence de Gray,

Soit $b = 110$. On applique l'inversion du code de Gray :

1. On a $g_2 = b_2 = 1$ (le bit de poids fort reste inchangé)
2. Puis, de façon récursive :
   - $g_1 = g_2 \oplus b_1 = 1 \oplus 1 = 0$
   - $g_0 = g_1 \oplus b_0 = 0 \oplus 0 = 0$

On obtient donc $g = 100$, qui correspond à l'entier décimal 4 en binaire.

Le mot `110` est le **5 élément** (rang 4, en commençant à 0) de la séquence de Gray de dimension 3 :

$$[000, 001, 011, 010, \mathbf{110}, 111, 101, 100]$$

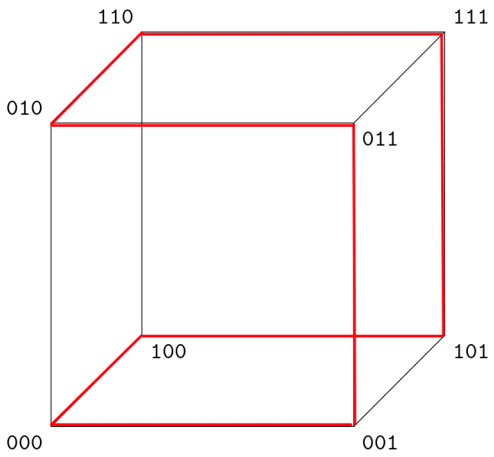

Figure 3: Cycle hamiltonien du code de Gray représenté en rouge sur l'hypercube C_3

Structure hamiltonienne

Le parcours du code de Gray sur l'hypercube C_n réalise un cycle hamiltonien : chaque sommet, correspondant à un mot binaire de longueur n, est visité exactement une fois, et chaque étape du parcours consiste en la modification d'un unique bit. Sur la Figure 3, le cycle hamiltonien induit par le code de Gray est représenté en rouge, illustrant la séquence de transitions élémentaires qui relie l'ensemble des sommets de l'hypercube pour $n = 3$ selon la construction explicitée dans l'exemple.

2.1.3 Premières observations et cadre théorique.

Les deux procédures présentées — Trotter-Johnson pour les permutations et code de Gray pour les mots binaires — reposent sur la construction de cycles hamiltoniens dans des graphes combinatoires : le graphe de Cayley pour S_n, et l'hypercube pour les mots binaires.

Cette propriété est le point de départ de la réflexion : elle garantit à la fois une énumération exhaustive, la possibilité de ranking/unranking en temps polynomial, et surtout on a la premières observations :

Observation 1 : Les codes de Gray et de Trotter-Johnson proposent chacun une énumération dans laquelle deux éléments consécutifs sont toujours voisins au sens minimal : dans le code de Gray, cela correspond à un changement d'un seul bit, tandis que dans l'algorithme de Trotter-Johnson, il s'agit d'une transposition.

Cette observation revêt une importance particulière, car elle met en évidence une régularité intrinsèque entre chaque rang, tant au niveau des labels qu'au niveau des solutions. Cette structure régulière constitue un fondement solide pour la conception de technique d'énumération efficaces et robustes, exploitables par l'algorithme maître M.

Remarque 1 : L'étude approfondie des algorithmes d'énumération assurant que deux éléments consécutifs diffèrent selon un critère combinatoire minimal (tel qu'une transposition ou un changement de bit) constitue un champ de recherche particulièrement dynamique en mathématiques discrètes et combinatoire algorithmique. Dans la littérature, on parle de **codes de Gray combinatoires** pour désigner l'ensemble des méthodes généralisant le principe du code de Gray à des structures combinatoires variées, au-delà des simples mots binaires. La synthèse récente et exhaustive [1] propose un vaste état de l'art sur les codes de Gray combinatoires, englobant notamment les algorithmes de Trotter-Johnson pour les permutations, les codes de Gray classiques, ainsi que de nombreux autres schémas d'énumération adaptés à des structures complexes (ensembles, partitions, arbres, etc.).

À ce stade, on commence à introduire le cadre théorique mis en place pour modéliser et comprendre ces techniques d'énumération, et c'est sous cet angle que sont présentés et détaillés les résultats obtenus concernant la construction d'une notion de correspondance entre ces énumérations, qui est en fait la formalisation de **Observation 1** et constitue le résultat principal des travaux réalisés.

2.1.4 Construction d'une énumération pour les mots du groupe symétrique S_n.

Proposition 1 : Sur le graphe de Cayley d'un groupe fini de cardinal n, construit à partir d'un ensemble de générateurs $\sigma_1, \dots, \sigma_k$, Tout cycle hamiltonien peut être décrit comme une séquence ordonnée d'applications successives de ces générateurs sur un élément initial x_0.

Preuve Proposition 1 :

La démonstration repose sur la structure même du graphe de Cayley associé au groupe fini considéré. Par définition, un graphe de Cayley $\mathrm{Cay}(G, \Sigma)$ est construit à partir d'un groupe G et d'un ensemble de générateurs $\Sigma = \{\sigma_1, \dots, \sigma_k\}$: chaque sommet du graphe correspond à un élément du groupe, et une arête relie deux sommets g et h s i et seulement si $h = \sigma_j \cdot g$ pour un certain générateur $\sigma_j \in \Sigma$.

Un cycle hamiltonien dans ce graphe est une suite $(x_1, \dots, x_n, x_n = x_0)$ telle que chaque x_{i+1} s'obtient de x_i par multiplication à gauche par un générateur, et que chaque élément du groupe apparaît exactement une fois dans la séquence. Ainsi, il existe une suite d'indices $(j_1, j_2, \dots, j_n)$ telle que

$$x_{i+1} = \sigma_{j_{i+1}} \cdot x_i$$

pour tout $i = 1, \dots, n-1$, avec x_0 fixé.

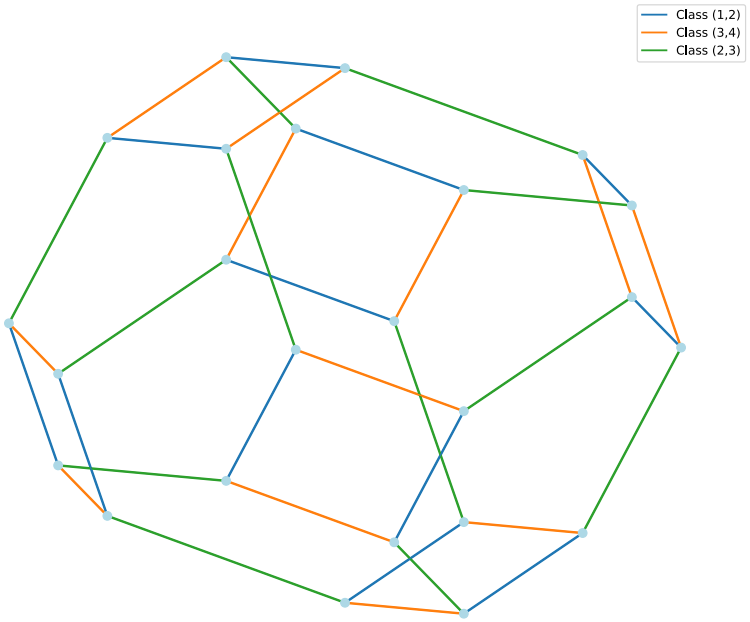

Figure 4: Le graphe de Cayley du groupe S_4 pour les transpositions adjacentes

Il s'agit du même graphe que la Figure 2 , cette fois-ci avec les arêtes colorées selon le générateur correspondant. Chaque couleur distingue l'action d'une transposition adjacente particulière, ce qui permet de visualiser explicitement la structure du graphe et la dynamique induite par la séquence des générateurs dans la procédure d'énumération.

La construction d'un cycle hamiltonien sur le graphe de Cayley s'effectue par l'application séquentielle des générateurs, suivant l'ordre imposé par la suite $(j_1, \dots, j_n)$. Cette procédure induit une énumération ordonnée des éléments du groupe : chaque élément est obtenu par une composition successive des générateurs **à partir d'un élément initial x_0.**

Définition 3 : On désigne une telle suite de générateurs comme une **procédure d'énumération du groupe pour l'ensemble de générateurs** Σ.

Cette notion permet de formaliser le parcours hamiltonien sous un angle algébrique et algorithmique, établissant ainsi une correspondance explicite entre la structure du groupe et la dynamique de l'énumération.

Remarque 2 : Cette perspective est largement adoptée dans la littérature mathématique consacrée à l'étude des cycles hamiltoniens dans les graphes de Cayley. Le travail de Schupp [2] constitue une référence de grande influence dans les travaux présenté ici : il y analyse la construction de cycles hamiltoniens pour les graphes de Cayley associés à certains groupes modulaires, en fournissant des exemples explicites de séquences de générateurs produisant de

tels cycles, ainsi que des résultats structurels sur ces séquences. Ce champ de recherche demeure particulièrement actif ; de nombreuses questions restent ouvertes concernant l'existence et la classification des cycles hamiltoniens pour des familles de groupes plus générales. Pour une synthèse des avancées et des problématiques , on pourra consulter la revue [8], qui propose un panorama des résultats connus sur les parcours hamiltoniens dans les graphes de Cayley, et met en lumière les directions de recherche encore inexplorées.

Proposition 2 : L'algorithme (présenté en 2.1.1) de Trotter-Johnson pour $n = 4$ produit exactement la même énumération que la procédure d'énumération :

$$[\ (3,4),\ (1,2),\ (2,3),\ (3,4),\ (1,2),\ (3,4),\ (2,3),\ (1,2),\ ]$$
$$[\ (3,4),\ (1,2),\ (2,3),\ (3,4),\ (1,2),\ (3,4),\ (2,3),\ (1,2),\ ]$$
$$[\ (3,4),\ (1,2),\ (2,3),\ (3,4),\ (1,2),\ (3,4),\ (2,3),\ (1,2),\ ]$$

Du groupe S_4 avec les générateurs $\Sigma_{\text{adj}} = \{(1\ 2),\ (2\ 3),\ (3\ 4)\}$, et en prenant comme point de départ $x_0 = (4, 3, 1, 2)$

Preuve de la Proposition 2 : La démonstration repose sur une construction explicite directe présentée en Annexe.

Un premier point à souligner est que l'on observe clairement la traduction du mouvement gauche $\rightarrow$ droite de l'algorithme dans la procédure d'énumération associée : celle-ci est constituée du segment

$$[\ (3,4),\ (1,2),\ (2,3),\ (3,4),\ (1,2),\ (3,4),\ (2,3),\ (1,2),\ ]$$

, répété trois fois.

Définition 4 : On désigne ce segment récurrent comme le **motif** (ici de taille $T = 8$) de la procédure d'énumération associée.

Théorème 1 : il existe au moins 16 motifs distincts de taille 8 engendrant une séquence valide pour S_4 avec les générateurs Σ_{adj}

Preuve Theoreme 1 : Construction explicite directe présentée en Annexe.

Ce résultat, qui peut sembler anodin, revêt pourtant une importance fondamentale : la mécanique de Trotter-Johnson correspond à la réinterprétation d'un des 16 chemins hamiltoniens possibles sur le graphe de Cayley qui lui est associé.

Remarque 3 : Si l'on considère une technique d'énumération fondée sur l'algorithme de Trotter-Johnson, plusieurs problématiques émergent alors. Il convient notamment d'interroger la notion d'optimalité du chemin hamiltonien retenu : existe-t-il des critères intrinsèques permettant de qualifier un parcours comme optimal au regard de la structure du graphe de Cayley ou des propriétés algorithmiques recherchées ? Par ailleurs, la question de l'équivalence entre

les différents cycles hamiltoniens se pose : Tous ces cycles sont-ils comparables en termes de régularité structurelle, ou d'impact sur la performance des algorithmes d'optimisation ?

Ces interrogations, loin d'être anecdotiques, soulèvent la nécessité de formaliser la notion d'**isomorphisme d'énumération**, c'est-à-dire de caractériser les conditions sous lesquelles deux procédures d'énumération peuvent être considérées comme structurellement équivalentes, et d'étudier les conséquences de cette équivalence sur la robustesse et l'efficacité des algorithmes développés. La suite du présent travail mettra en évidence que la prise en compte rigoureuse de ces aspects est déterminante pour l'élaboration de méthodes efficaces.

2.1.5 Construction d'une énumération pour les mots binaires du cube C_n.

On l'a vu en partie 2.1.3, le code de Gray constitue également une technique d'énumération fondée sur l'existence d'un cycle hamiltonien dans un graphe particulier, à savoir l'hypercube.

Une question naturelle qui se pose alors est de déterminer si ce graphe peut lui aussi être interprété comme un graphe de Cayley d'un groupe approprié, pour un certain choix de générateurs.

Proposition 3 : Soit $C_n = \{(x_1, x_2, \dots, x_n) \mid x_i \in \{-1, 1\}\}$ muni de la **multiplication composante par composante**. Ce groupe est abélien, d'ordre 2^n, et isomorphe à $\mathbb{Z}_2^n$ via l'identification $-1 \leftrightarrow 0 \in \mathbb{Z}_2$ et $1 \leftrightarrow 1$. On considère l'ensemble de générateurs :

$$\Sigma_{\text{flip}} = \{\sigma_i \mid 1 \leq i \leq n\}$$

où chaque générateur $\sigma_i \in C_n$ est défini par :

$$\sigma_i = (1, \dots, 1, \underset{i}{-1}, 1, \dots, 1)$$

Autrement dit, σ_i est l'élément de C_n qui a -1 en position i, et 1 ailleurs. Pour tout $x \in C_n$, la multiplication $\sigma_i \cdot x$ correspond à **changer le signe de la i-ème coordonnée de x**.

Le **graphe de Cayley** de C_n associé à Σ_{flip} est alors isomorphe à l'**hypercube** C_n :

- chaque sommet correspond à un élément $x \in C_n$,
- deux sommets sont reliés par une arête s'ils diffèrent sur **exactement une coordonnée**.

Proposition 4 : Le code de Gray sur C_3 proposé en partie 2.1.3 correspond à une procédure d'énumération obtenue par la séquence ordonnée des générateurs $\Sigma_{\text{flip}} = \{\sigma_1, \sigma_2, \sigma_3\}$, où chaque σ_i réalise une invertion du i-ème bit. En prenant comme point de départ $x_0 = (1, -1, -1)$, la procédure d'énumération associée au code de Gray est alors :

$$[\sigma_3,\ \sigma_2,\ \sigma_3,\ \sigma_1,\ \sigma_3,\ \sigma_2,\ \sigma_3,\ \sigma_1]$$

On obtient ainsi une séquence explicite des éléments du groupe, à savoir :

$$(-1,-1,-1),\ (-1,-1,1),\ (-1,1,1),\ (-1,1,-1),\ (1,1,-1),\ (1,1,1),\ (1,-1,1),\ (1,-1,-1)$$

laquelle s'identifie naturellement, par l'isomorphisme précédemment établi, à la suite des mots binaires :

$$(0,0,0),\ (0,0,1),\ (0,1,1),\ (0,1,0),\ (1,1,0),\ (1,1,1),\ (1,0,1),\ (1,0,0)$$

qui coincide bien avec l'énumération de l'exemple vue en 2.1.3.

Théorème 2 : Il existe exactement 12 séquences valides pour C_3 avec les générateurs Σ_{flip}

Preuve Theoreme 2: Construction explicite directe présentée en Annexe.

2.1.6 Isomorphismes de représentation opératoires entre groupes, graphs et cycle

On a mis en évidence que les deux algorithmes d'énumération étudiés — à savoir l'algorithme de Trotter-Johnson pour les permutations et le code de Gray pour les mots binaires — peuvent être interprétés comme des parcours hamiltoniens sur des graphes associés à des groupes finis : respectivement, le graphe de Cayley du groupe symétrique S_n pour les permutations, et l'hypercube (lui-même graphe de Cayley du groupe abélien $\mathbb{Z}_2^n$) pour les mots binaires.

Cette lecture sous l'angle de la théorie des groupes révèle un point fondamental : un même ensemble de générateurs (opérations élémentaires du groupe, telles que les transpositions adjacentes ou les invertions de bits) permet de décrire une multitude de cycles hamiltoniens distincts sur le graphe de Cayley considéré. **Autrement dit, il existe une pluralité de parcours exhaustifs de l'espace, chacun correspondant à une séquence particulière d'applications des générateurs, mais tous partageant la propriété structurelle d'être obtenus à partir des mêmes opérations fondamentales.**

Des représentations linéaires aux représentations opératoires

La théorie des représentations d'un groupe G consiste à étudier des morphismes $\rho : G \to \mathrm{GL}(V)$, où V est un espace vectoriel. Ce cadre, essentiellement linéaire, permet de comprendre les symétries de G à travers la structure algébrique de V.

Exemple : la représentation du groupe S_4 comme groupe d'isométries du cube

Le groupe S_4, peut être représenté comme un sous-groupe du groupe orthogonal $O(3)$, via son action sur les 4 diagonales principales d'un cube centré à l'origine.

Concrètement, soit $V = \mathbb{R}^3$, l'espace euclidien tridimensionnel. Il existe une représentation linéaire

$$\rho : S_4 \to \mathrm{GL}(V)$$

dont l'image est contenue dans $O(3)$, telle que pour toute permutation $\sigma \in S_4$, l'opérateur $\rho(\sigma)$ est une isométrie du cube qui permute ses diagonales principales selon σ.

Autrement dit, on fait agir les éléments de S_4 comme transformations géométriques du cube, et l'action est fidèle. **Ce type de représentation donne une incarnation géométrique d'un groupe abstrait, ce qui est un des thèmes classiques de la théorie des représentations.**

Observation 3 : Il convient de noter que cet exemple n'est pas anodin : Traditionnellement, la théorie des représentations pour les groupes finis permet d'adopter une perspective géométrique, en étudiant les actions de groupes à travers leurs propriétés intrinsèques dans le cadre des espaces vectoriels. Pour une compréhension approfondie de ces concepts, on pourra consulter le cours de référence [3], qui offre une présentation structurée et accessible du sujet.

On propose ici en fait une généralisation conceptuelle inspiré du formalisme des représentations, en remplaçant **l'action de ces générateur sur un espace vectoriel** par **l'action de ces procédures sur un espace d'etats**.

Procédures opératoires

De façon générale, soit G un groupe fini, et soit $\Sigma = \{\sigma_1, \dots, \sigma_k\} \subset G$ un ensemble de générateurs (qui ne sont pas nécessairement symétriques, minimaux).

Une **Procédure opératoire** $w \in \Sigma^*$ est une suite finie ordonnée de symboles concaténés :

$$w = \sigma_{i_1}\sigma_{i_2}\cdots\sigma_{i_\ell} \in \Sigma^*, \tag{2}$$

où Σ^* désigne le monoïde engendré par Σ, c'est-à-dire essentiellement l'ensemble des mots qu'il est possible de créer par concaténation des élément de Σ.

Remarque 4: D'un point de vue fondamental, les procédures opératoires peuvent être assimilées à des mots sur l'alphabet des générateurs, c'est-à-dire à des éléments du monoïde Σ^*. Néanmoins, il est important de s'éloigner ici du vocabulaire usuel de la théorie des langages formels, afin de mieux mettre en évidence la spécificité du cadre des représentations opératoires. Ce qui importe avant tout, c'est la lecture séquentielle d'un mot $w \in \Sigma^*$ de gauche à droite, interprétée comme une suite d'instructions à exécuter sur l'espace de solution à parcourir.

Définition 5 — Représentation opératoire d'un groupe

Soit E un ensemble fini (appelé **espace de symboles** ou **espace d'états**).

On appelle **représentation opératoire** de G sur E la donnée d'une **action opératoire** de ces générateurs sur E, qui est un morphisme de monoïdes :

$$\pi : \Sigma^* \to \mathrm{Part}(E), \tag{3}$$

tel que :

- Σ^* est le monoïde engendré par Σ (les mots sur Σ par concaténation),
- $\mathrm{Part}(E)$ désigne l'ensemble des applications partielles $f : E \rightharpoonup E$,
- Chaque générateurs $\sigma \in \Sigma$ est envoyée sur une transformation partielle $\pi(\sigma)$,
- $\pi(w)$ est défini pour tout mot $w = \sigma_{i_1}\cdots\sigma_{i_k} \in \Sigma^*$ par :

$$\pi(w) := \pi(\sigma_{i_1}) \circ \cdots \circ \pi(\sigma_{i_k}), \tag{4}$$

lue de gauche à droite, où $\circ$ est la composition usuelle de fonctions.

Les représentations opératoires visent à modéliser la dynamique effective induite par l'exécution d'une procédure opératoire w. Dès lors qu'un algorithme d'énumération peut être interprété comme la réalisation d'une telle procédure, il devient possible d'en construire la représentation opératoire correspondante,

afin de l'analyser ou de le comparer à d'autres algorithmes d'énumération dans un cadre unifié.

Définition 6 : La suite des points visités depuis un état initial $e_0 \in E$ par une procédure opératoire $w \in \Sigma^*$, est appelée **trajectoire opératoire** de w depuis e_0. Si cette trajectoire parcourt E tout entier, on dit qu'elle est **couvrante**.

Remarque 5 :

L'introduction d'actions partielles permet de modéliser de manière naturelle les contraintes intrinsèques à chaque structure.

Par exemple, si on considère le cas du calcul quantique avec portes conditionnelles :

Un algorithme quantique dans lequel une porte σ n'est appliquée que si le premier qubit est dans l'état $|1\rangle$.

- Espace d'états E : ensemble des états d'un registre quantique.
- Action partielle : $\pi(\sigma)(e)$ n'est définie que si le premier qubit de e est $|1\rangle$.

Ce formalisme permet ainsi de capturer précisément les contraintes physiques ou logiques imposées par l'architecture quantique (par exemple, l'activation conditionnelle d'une porte comme CNOT), et de les intégrer de façon rigoureuse dans la modélisation opératoire.

Remarque 6 : **La proposition 2 constitue une illustration explicite de trajectoire opératoire couvrante** au sens défini précédemment. Dans ce cas, l'ensemble E est formé des (24) 4-uplets obtenus par permutation des entiers $1, 2, 3, 4$; les générateurs employés correspondent aux transpositions agissant sur des positions adjacentes, et la procédure d'énumération encode la séquence ordonnée d'applications de ces générateurs à partir de l'état initial $(1, 2, 3, 4)$. À chaque itération, l'état courant de E est transformé conformément à la dynamique prescrite par la représentation opératoire, ce qui fournit une concrétisation rigoureuse du cadre formel introduit et met en évidence la structure algorithmique sous-jacente à l'énumération considérée.

Définition 7 — Isomorphisme de représentations opératoires

Soient E et E' deux espace d'états de même carinalité. Soient (G, Σ, E, π) et (G', Σ', E', π') deux représentations opératoires.

On dit qu'il existe un **isomorphisme de représentations opératoires** entre les deux systèmes s'il existe :

- une bijection $\varphi : \Sigma \to \Sigma'$,
- une bijection $\theta : E \to E'$, telles que pour tout mot $w = \sigma_{i_1} \cdots \sigma_{i_k} \in \Sigma^*$, on ait :

$$\theta(\pi(w)(\cdot)) = \pi'(\varphi(w))(\theta(\cdot)) \tag{5}$$

dans le sens des applications partielles (définies là où les deux côtés le sont), **où $\varphi(w)$ est un raccourci de notation pour désigner le mot $\varphi(\sigma_{i_1}) \cdots \varphi(\sigma_{i_k}) \in \Sigma'^*$.**

Cet isomorphisme (φ, θ) exprime la compatibilité complète entre les deux représentations opératoires : les mots sur Σ^* et ceux sur Σ'^* ont exactement le même effet séquentiel, via θ, sur leurs espaces d'états respectifs.

Définition 8: On peut également considérer la notion **d'isomorphisme partiel** de représentations opératoires, c'est-à-dire **un isomorphisme restreint à un sous-ensemble de Σ^*.** Autrement dit, il s'agit de caractériser une équivalence entre deux représentations opératoires, mais uniquement sur une ou plusieurs procédures opératoires spécifiques (i.e., sur un sous-ensemble de mots de Σ^*), et non nécessairement sur l'ensemble complet des mots. Cette notion permet d'étudier des correspondances structurelles pour certains algorithmes particuliers, **notamment dans un cadre où l'on cherche souvent à implémenter une seule de ces correspondances.**

Définition 9 — Isomorphisme partiel de représentations opératoires périodique

Soient E et E' deux ensembles finis, possiblement de cardinalités distinctes a priori.

On dira qu'il existe un isomorphisme partiel opératoire **périodique** entre deux représentations opératoires

$$(G, \Sigma, E, \pi) \quad \text{et} \quad (G', \Sigma', E', \pi'),$$

s'il existe :

- au moins une procédure opératoire $w \in \Sigma^*$ et une procédure opératoire $w' \in (\Sigma')^*$,
- un entier $m \geq 1$,
- une bijection de générateurs $\varphi : \Sigma \to \Sigma'$ telle que

$$\varphi(w^m) = w', \tag{6}$$

- et une bijection d'états

$$\theta : E \to E' \times \mathbb{Z}/m\mathbb{Z}$$

telle que, pour tout $j \in \{0, \dots, m \cdot |w| - 1\}$,

$$\theta(e_j) = (e'_j, \text{bloc}(j)), \tag{7}$$

où $\text{bloc}(j) \in \mathbb{Z}/m\mathbb{Z}$ désigne l'indice de répétition du mot w (c'est-à-dire $\text{bloc}(j) = \left\lfloor \frac{j}{|w|} \right\rfloor \bmod m$),

et que la compatibilité d'action suivante soit vérifiée pour tout e_0 :

$$\theta(\pi(w^m)(e_0)) = \pi'(w')(\theta(e_0)). \tag{8}$$

Cette definition permet de comparer des groupes de tailles différentes, sur des espace d'état de dimention différentes, via une **procédure opératoire commune**.

2.1.7 Construction

On présente ici un exemple explicite d'**isomorphisme partiel de représentations opératoires périodiques** entre le groupe symétrique S_4 (agissant sur les permutations de 4 éléments) et le groupe C_3 (agissant sur le cube de dimension 3), illustrant la correspondance structurelle entre :

- une procédure de type **Trotter–Johnson** sur S_4,
- et une procédure de type **code de Gray** sur C_3.

Proposition 5: Il existe une procédure opératoire permettant d'établir un **isomorphisme partiel de représentations opératoires périodique** entre le groupe S_4, agissant sur les 24 permutations de $(1, 2, 3, 4)$ par transpositions adjacentes, et le groupe C_3, agissant sur l'ensemble des états du cube par inversion de bits.

Groupe S_4

- L'espace d'états $E = \{(i_1, i_2, i_3, i_4) \mid (i_1, i_2, i_3, i_4)$ permutation de $(1, 2, 3, 4)\}$
- Générateurs : $\Sigma = \{\tau_1 = (1\,2),\ \tau_2 = (2\,3),\ \tau_3 = (3\,4)\}$, où chaque τ_i est la permutation qui échange les positions i et $i + 1$ et l'action opératoire π est est donnée par

$$\pi(\tau_1) = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \pi(\tau_2) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \pi(\tau_3) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

Et π se compose par multiplication matricielle à gauche sur les vecteurs colonnes représentant les points de E.

Groupe C_3

- On considère les générateurs $\Sigma' = \{\sigma_1, \sigma_2, \sigma_3\}$, définis comme suit :

$$\pi'(\sigma_1) = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \pi'(\sigma_2) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}, \quad \pi'(\sigma_3) = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

- L'espace d'états $E' = \{-1, 1\}^3 = \{(i_1, i_2, i_3) \mid i_1, i_2, i_3 \in \{-1, 1\}\}$, et l'action est donnée par multiplication matricielle à gauche sur les vecteurs colonnes représentant les points de E'.

On définit donc formellement les représentations opératoires suivantes :

Sur S_4 :

(G, Σ, E, π), avec :

- $G = S_4$,
- $\Sigma = \{\tau_1, \tau_2, \tau_3\}$,
- $E = \{(i_1, i_2, i_3, i_4) \mid (i_1, i_2, i_3, i_4)$ permutation de $(1, 2, 3, 4)\}$,
- π : action par multiplication matricielle a gauche.

Sur C_3 :

(G', Σ', E', π'), avec :

- $G' = C_3$,
- $\Sigma' = \{\sigma_1, \sigma_2, \sigma_3\}$,
- $E' = \{-1, 1\}^3$,
- π' : action par multiplication matricielle a gauche.

Procédure opératoire couvrante de C_3 pour Σ'

Soit la procédure opératoire :

$$\mathcal{W}_{C_3} = (\sigma_1,\ \sigma_2,\ \sigma_3,\ \sigma_2,\ \sigma_3,\ \sigma_2,\ \sigma_1,\ \sigma_2), \tag{9}$$

appliquée à l'état initial $e_0 = (-1, -1, -1)$. On obtient la trajectoire couvrante suivante sur E' :

$$
\begin{aligned}
(-1,-1,-1) &\xrightarrow{\sigma_1} (-1,1,-1)\\
&\xrightarrow{\sigma_2} (-1,1,1)\\
&\xrightarrow{\sigma_3} (1,1,1)\\
&\xrightarrow{\sigma_2} (1,1,-1)\\
&\xrightarrow{\sigma_3} (1,-1,-1)\\
&\xrightarrow{\sigma_2} (1,-1,1)\\
&\xrightarrow{\sigma_1} (-1,-1,1)\\
&\xrightarrow{\sigma_2} (-1,-1,-1)
\end{aligned}
$$

On définit un isomorphisme identidé sur les générateurs :

$$\varphi : \Sigma' \to \Sigma, \quad \sigma_i \mapsto \tau_i.$$

On définit l'espace augmenté $E' \times \mathbb{Z}/3\mathbb{Z}$, où l'indice de répétition du mot est donné par :

$$\mathrm{bloc}(j) = \left\lfloor \frac{j}{8} \right\rfloor \bmod 3.$$

On construit une bijection :

$$\theta : E \to E' \times \mathbb{Z}/3\mathbb{Z}, \qquad \theta(e_j) = (e'_j,\, \mathrm{bloc}(j)),$$

où $e'_j \in E'$ est l'état visité à l'instant $j \bmod 8$ par la procédure d'énumération de C_3.

La procédure $\mathcal{W}_{C_3}$, répétée trois fois, donne une séquence de 24 éléments sur E'. En projetant chaque bloc de 8 états via θ, et en appliquant φ à la procédure, on obtient une procédure $\mathcal{W}_{S_4}$ sur E. En partant de l'état initial $(1,2,3,4)$, la trajectoire opératoire générée par cette procédure s'écrit explicitement en Table 1.

Pour accompagner la lecture de cette table, la Figure 5 présente une visualisation structurée : elle représente les parcours successifs des trois blocs (rouge, bleu, vert) sur chacun des cubes, puis leur projection sur le graphe de Cayley de S_4. Cette figure met en évidence, d'une part, le cycle hamiltonien généré par la trajectoire opératoire, et d'autre part, la périodicité induite par la répétition de la procédure, chaque couleur correspondant à un bloc distinct dans la séquence opératoire.

Une analyse approfondie de la construction de cet isomorphisme de représentations opératoires, ainsi que des justifications détaillées concernant le choix de l'action π', est présentée en annexe. Le lecteur est invité à s'y référer pour une discussion sur la sélection des matrices associées aux générateurs.

Table 1: Tableau de correspondance opératoire entre S_4 et C_3

Étape	**État dans S_4**	$\mathcal{W}_{S_4}$	**État dans C_3 (bloc k)**	$\mathcal{W}_{C_3}$
0	(1,2,3,4)	–	$(-1,-1,-1)$ ()	–
1	(2,1,3,4)	τ_1	$(-1,1,-1)$ (1)	σ_1
2	(2,3,1,4)	τ_2	$(-1,1,1)$ (1)	σ_2
3	(2,3,4,1)	τ_3	$(1,1,1)$ (1)	σ_3
4	(2,4,3,1)	τ_2	$(1,1,-1)$ (1)	σ_2
5	(2,4,1,3)	τ_3	$(1,-1,-1)$ (1)	σ_3
6	(2,1,4,3)	τ_2	$(1,-1,1)$ (1)	σ_2
7	(1,2,4,3)	τ_1	$(-1,-1,1)$ (1)	σ_1
8	(1,4,2,3)	τ_2	$(-1,-1,-1)$ (2)	σ_2
9	(4,1,2,3)	τ_1	$(-1,1,-1)$ (2)	σ_1
10	(4,2,1,3)	τ_2	$(-1,1,1)$ (2)	σ_2
11	(4,2,3,1)	τ_3	$(1,1,1)$ (2)	σ_3
12	(4,3,2,1)	τ_2	$(1,1,-1)$ (2)	σ_2
13	(4,3,1,2)	τ_3	$(1,-1,-1)$ (2)	σ_3
14	(4,1,3,2)	τ_2	$(1,-1,1)$ (2)	σ_2
15	(1,4,3,2)	τ_1	$(-1,-1,1)$ (2)	σ_1
16	(1,3,4,2)	τ_2	$(-1,-1,-1)$ (3)	σ_2
17	(3,1,4,2)	τ_1	$(-1,1,-1)$ (3)	σ_1
18	(3,4,1,2)	τ_2	$(-1,1,1)$ (3)	σ_2
19	(3,4,2,1)	τ_3	$(1,1,1)$ (3)	σ_3
20	(3,2,4,1)	τ_2	$(1,1,-1)$ (3)	σ_2
21	(3,2,1,4)	τ_3	$(1,-1,-1)$ (3)	σ_3
22	(3,1,2,4)	τ_2	$(1,-1,1)$ (3)	σ_2
23	(1,3,2,4)	τ_1	$(-1,-1,1)$ (3)	σ_1
24	(1,2,3,4)	τ_2	$(-1,-1,-1)$ (1)	σ_2

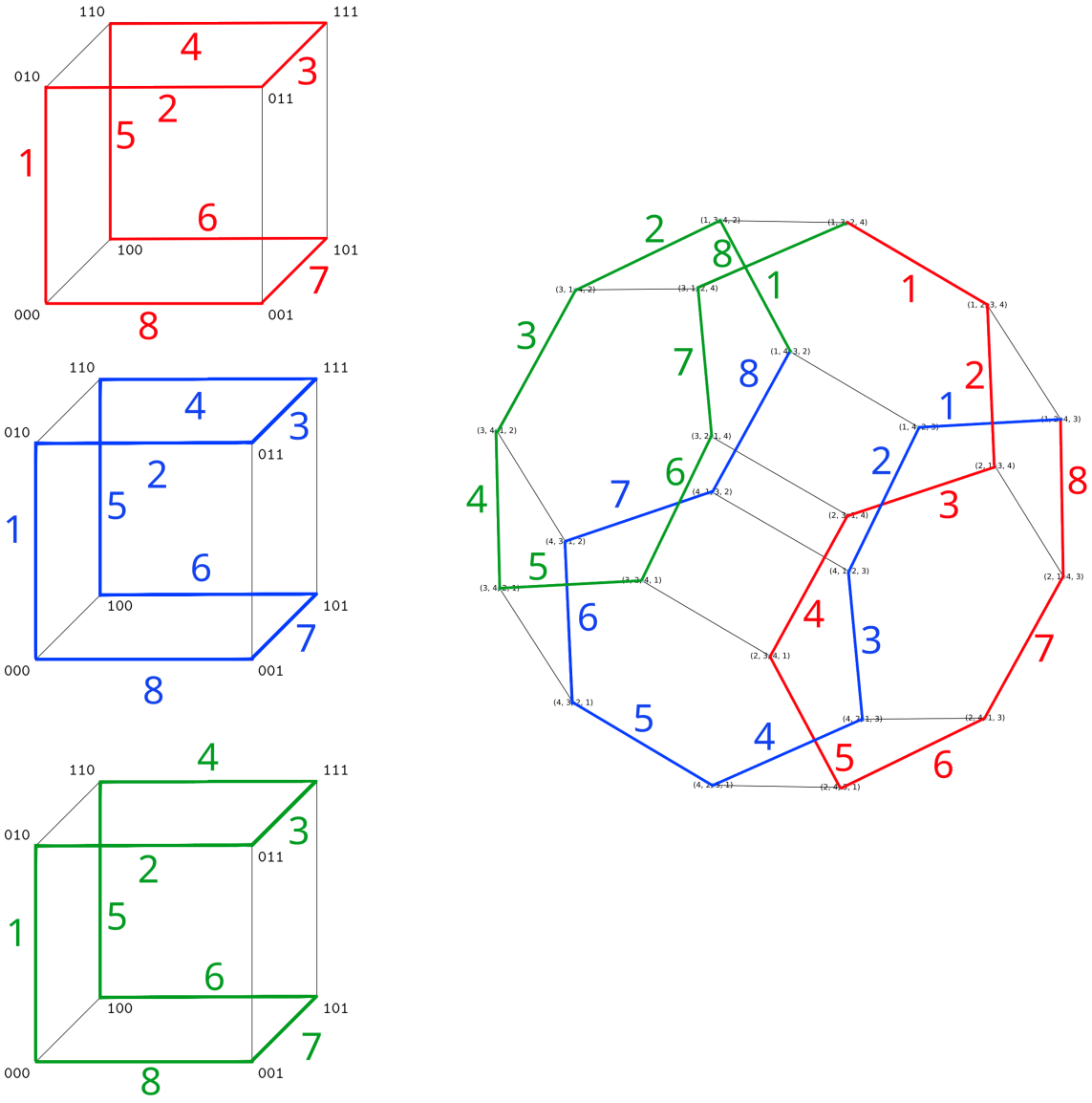

Figure 5: Illustration de l'isomorphisme de représentation opératoire décris en Table 1, en rouge le bloc(1), en bleu le bloc(2), et en vert le bloc(3).

On constate que chaque élément de E apparaît exactement une fois dans cette séquence, ce qui confirme que la procédure opératoire $\mathcal{W}_{S_4}$ est bien convrante pour la représentation opératoire (G, Σ, E, π) qui conciste à réalise un cycle hamiltonien sur le graphe de Cayley associé.

2.2 : Construction explicite d'une correspondance entre $C^*_{n'}$ et S_n

Dans cette section, on propose une construction générale permettant, pour tout entier n, de définir une famille de représentations opératoires agissant sur un espace de mots binaires de cardinal arbitraire. **L'objectif est de formaliser l'intégration de la périodicité précédemment établie, en élaborant une construction systématique permettant d'incorporer l'indexation par blocs au sein même de la structure binaire de l'espace d'états.**

Plus précisément, il s'agit de construire explicitement un espace augmenté E qui demeure un espace de mots binaires, tout en incorporant la structure périodique nécessaire à la définition 14. Par exemple, dans le cas de la partie 2.1.7 où on souhaite représenter 3 blocs binaires de 8 éléments (soit $8 \times 3 = 24$ états), il est possible de les encoder dans un espace binaire de dimension 5, de façon à préserver la compatibilité avec la définition d'isomorphisme périodique.

L'objectif de cette construction est de garantir que l'algorithme maître M, qui opère sur l'ensemble complet des labels binaires, puisse interagir de manière cohérente et systématique avec le circuit quantique via un isomorphisme opératoire. Il s'agit donc de s'assurer que cet isomorphisme soit bien défini sur l'intégralité de l'espace des labels, de sorte que toute requête de M puisse être traitée sans ambiguïté et sans perte d'information structurelle, assurant ainsi une communication fiable entre le système quantique et l'algorithme classique.

Remarque 7: Dans la suite, on considérera systématiquement des procédure opératoires couvrantes, sans exiger qu'elles soient nécessairement cycliques. Cette généralisation, qui consiste à passer d'un cycle hamiltonien à un simple chemin hamiltonien, s'avère naturelle dans le cadre des représentations opératoires : en effet, celles-ci sont définies sur l'ensemble des mots de l'alphabet Σ^*, et non exclusivement sur procédure opératoires couvrantes ou cyclique. Cette approche permet de construire plus aisément des structures combinatoires pertinentes et adaptées aux besoins de l'optimisation. On verra dans la section consacrée aux généralisations que cette relaxation possède une signification particulière pour les types de représentations opératoires étudiés, et qu'elle ouvre la voie à des constructions plus flexibles et efficaces dans le contexte du TSP.

2.2.1 Contexte et Notations

On considère :

- $C_{n'} = \{0,1\}^{n'}$, l'ensemble des labels binaires de dimension n', contenant $2^{n'}$ éléments.

Pour établir une correspondance entre S_n et $C_{n'}$, il est nécessaire de restreindre $C_{n'}$ à un sous-ensemble $C^*_{n'}$ tel que :

- $|C^*_{n'}| = |S_n| = n!$.
- Cette restriction respecte certaines contraintes structurelles, rendant la construction non triviale.

Définition de $C^*_{n'}$

Soit $n' = \lceil \log_2(n!) \rceil$, la dimension minimale nécessaire pour représenter $n!$ permutations en binaire. Un vecteur $q \in C_{n'}$ est défini comme une suite de n' bits (0 ou 1).

Pour restreindre $C_{n'}$ à $C^*_{n'}$, une partition $I = (a_1, a_2, \ldots, a_k)$ de n' est introduite, où chaque a_i représente la taille d'un bloc de q, par exemple, si $q = (1, 0, 1, 0)$:

- Pour $I = (2, 2)$, alors $q_I = (1, 0)(1, 0)$.
- Pour $I = (1, 3)$, alors $q_I = (1)(0, 1, 0)$.

Remarque 8: Comme on cherche à minimiser le nombre de qubits dans le circuit quantique, il est essentiel de rendre l'encodage le plus compact possible. Pour cela, on cherche une partition I qui élimine le moins de tuples possibles, c'est-à-dire qui impose le minimum d'états interdits. On verra dans la partie implémentation comment il est parfois possible de concevoir des circuits très spécifiques capables de censurer certains états, rendant ainsi cette construction réalisable et très interressante en pratique.

Pour mieux comprendre cette construction, on examine deux exemples concrets pour le cas où $n = 3$.

Exemple : $I = (3)$

Le cas le plus simple est celui où $I = (3)$, c'est-à-dire que l'on considère un seul bloc de taille 3, la représentation canonique.

Pour $n = 3$, on a $n! = 6$ permutations et $n' = \lceil \log_2(6) \rceil = 3$. Ainsi, $C_{n'} = \{0, 1\}^3$ contient $2^3 = 8$ éléments.

Pour obtenir $C^*_{n'}$, il faut alors supprimer $2^3 - 6 = 2$ éléments de $C_{n'}$. Par exemple :

- $C_{n'} = \{(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)\}$.
- $C^*_{n'} = \{(0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0)\}$, où les tuples $(0, 0, 0)$ et $(1, 1, 1)$ ont été supprimés.

Exemple : $I = (1, 2)$

Si on choisit une partition $I = (1, 2)$, chaque vecteur $q \in C_{n'}$ est divisé en deux parties :

- Une première partie de taille 1.
- Une seconde partie de taille 2.

Ainsi, $C^*_{n'}$ devient :

- $C_{n'} = \{(0)(0,0), (0)(0,1), (0)(1,0), (0)(1,1), (1)(0,0), (1)(0,1), (1)(1,0), (1)(1,1)\}$.
- $C^*_{n'} = \{(0)(0,1), (0)(1,0), (0)(1,1), (1)(1,0), (1)(0,1), (1)(1,1)\}$.

Dans ce cas, seul le tuple $(0, 0)$ est interdit (dans la seconde partie) et qui a permis de supprimer 2 éléments de $C_{n'}$, à savoir $(0, 0, 0)$ et $(1, 0, 0)$.

En ce sens, $I = (1, 2)$ est ainsi une meilleure partition que $I = (3)$, puisque le nombre de tuples interdits est de 1 au lieu de 2. Le choix du tuple à supprimer est arbitraire. Par souci de simplicité, lorsque l'on doit exclure un unique tuple, on choisit systématiquement $(0, 0, \dots, 0)$. Cette convention est explicitement indiquée à chaque occurrence pour garantir la clarté de la construction.

2.2.2 Propriétés d'une partition optimale

Une première question naturelle est de savoir si l'on peut toujours trouver une partition I telle que le nombre de tuples interdits dans chaque partie soit au plus 1.

On peut observer dans la table Table 2 que, pour chaque valeur de n, la structure arithmétique de $2^n - 1$ et de $n!$ joue un rôle clé dans la possibilité de construire une partition optimale pour l'encodage binaire des permutations. Plus précisément, lorsque les facteurs premiers de $n!$ sont présents dans la décomposition de $2^n - 1$, il est possible de minimiser le nombre de tuples interdits dans chaque bloc, ce qui permet un encodage plus compact et efficace. À l'inverse, dès que certains facteurs premiers de $n!$ ne figurent pas dans $2^n - 1$, la construction optimale devient impossible et le nombre d'états à exclure augmente, rendant la préparation quantique plus complexe.

Table 2: Décomposition arithmétique de $2^n - 1$ et $n!$ pour $n = 1$ à 20

n	$2^n - 1$	Facteurs premiers de $2^n - 1$	$n!$	Facteurs premiers de $n!$
1	1	1	1	1
2	3	3	2	2
3	7	7	6	2×3
4	15	3×5	24	$2^3 \times 3$
5	31	31	120	$2^3 \times 3 \times 5$
6	63	$3^2 \times 7$	720	$2^4 \times 3^2 \times 5$
7	127	127	5040	$2^4 \times 3^2 \times 5 \times 7$
8	255	$3 \times 5 \times 17$	40320	$2^7 \times 3^2 \times 5 \times 7$
9	511	7×73	362880	$2^7 \times 3^4 \times 5 \times 7$
10	1023	$3 \times 11 \times 31$	3628800	$2^8 \times 3^4 \times 5^2 \times 7$
11	2047	23×89	39916800	$2^8 \times 3^4 \times 5^2 \times 7 \times 11$
12	4095	$3^2 \times 5 \times 7 \times 13$	479001600	$2^{10} \times 3^5 \times 5^2 \times 7 \times 11$
13	8191	8191	6227020800	$2^{10} \times 3^5 \times 5^2 \times 7 \times 11 \times 13$
14	16383	$3 \times 43 \times 127$	87178291200	$2^{11} \times 3^5 \times 5^2 \times 7^2 \times 11 \times 13$
15	32767	$7 \times 31 \times 151$	1307674368000	$2^{11} \times 3^6 \times 5^3 \times 7^2 \times 11 \times 13$
16	65535	$3 \times 5 \times 17 \times 257$	20922789888000	$2^{15} \times 3^6 \times 5^3 \times 7^2 \times 11 \times 13$
17	131071	131071	355687428096000	$2^{15} \times 3^6 \times 5^3 \times 7^2 \times 11 \times 13 \times 17$
18	262143	$3^3 \times 7 \times 19 \times 73$	6402373705728000	$2^{16} \times 3^8 \times 5^3 \times 7^2 \times 11 \times 13 \times 17$
19	524287	524287	121645100408832000	$2^{16} \times 3^8 \times 5^3 \times 7^2 \times 11 \times 13 \times 17 \times 19$
20	1048575	$3 \times 5^2 \times 11 \times 31 \times 41$	2432902008176640000	$2^{18} \times 3^8 \times 5^4 \times 7^2 \times 11 \times 13 \times 17 \times 19$

Table 3: Partitions optimales pour l'encodage binaire des permutations de $n = 3$ à 10

n	$n!$ (décomposition)	Partition optimale I	Facteurs premiers utilisés $(2^n - 1)$
3	$6 = 2 \times 3$	$(1, 2)$	$6 = 2 \times 3$
4	$24 = 2^3 \times 3$	$(3, 2)$	$24 = 8 \times 3$
5	$120 = 2^3 \times 3 \times 5$	$(3, 4)$	$120 = 8 \times 15$
6	$720 = 2^4 \times 3^2 \times 5$	$(4, 4, 2)$	$720 = 16 \times 15 \times 3$
7	$5040 = 2^4 \times 3^2 \times 5 \times 7$	$(4, 4, 3, 2)$	$5040 = 16 \times 15 \times 7 \times 3$
8	$40320 = 2^7 \times 3^2 \times 5 \times 7$	$(7, 4, 3, 2)$	$40320 = 128 \times 15 \times 7 \times 3$
9	$362880 = 2^7 \times 3^4 \times 5 \times 7$	$(7, 6, 4, 2)$	$362880 = 128 \times 63 \times 15 \times 3$
10	$3628800 = 2^8 \times 3^4 \times 5^2 \times 7$	$(8, 6, 4, 4)$	$3628800 = 256 \times 63 \times 15 \times 15$

La Table 3 résume pour chaque n de 3 à 10 :

- $n!$ et sa décomposition en facteurs premiers,
- La partition optimale I permettant de minimiser les tuples interdits,
- la décomposition du produit induite par cette partition.

Exemple : $n = 4, 5$

Pour $4! = 24 = 2^3 \times 3 = 8 \times 3$, la partition optimale est $I = (3, 2)$, car elle donne $(2^3 = 8,\ 2^2 - 1 = 3)$ avec un seul tuple interdit : $(0, 0)$.

Pour $5! = 120 = 2^3 \times 3 \times 5 = 8 \times 15$, la partition optimale est $I = (3, 4)$, car elle donne $(2^3 = 8,\ 2^4 - 1 = 15)$ avec un seul tuple interdit : $(0, 0, 0, 0)$.

Premier contre-exemple : $n = 11$

Pour $11! = 39916800 = 2^8 \times 3^4 \times 5^2 \times 7 \times 11$, on constate que 11 n'apparaît pas dans la décomposition en facteurs premiers de $2^n - 1$, sauf pour $2^{10} - 1 = 1023 = 3 \times 11 \times 31$. Cependant, 11! n'est pas divisible par 31.

Ainsi, pour toute valeur de $n \in [11, 30]$, il est impossible de trouver une partition I où le nombre de tuples interdits dans chaque partie est au plus 1, car 11 n'apparaît pas dans la décomposition en facteurs premiers de $2^n - 1$.

2.2.3 Une procédure opératoire sur $C^*_{n'}$

Dans la pratique, comme on l'observe, il n'est généralement pas possible de se contenter d'une restriction de la forme $2^n - 1$ pour tous les entiers n. Pour certains n, il faut considérer des sous-ensembles de la forme $2^n - k$, pour un certain k dépendant de la structure arithmétique de $n!$. Par exemple, pour 11!, l'encodage optimal requiert l'utilisation d'un bloc de 4 bits, soit 16 états,

parmi lesquels 5 doivent être explicitement exclus afin d'obtenir 11 états isolés et construire une correspondance exacte avec le cardinal de l'ensemble des solutions considérées.

Remarque 9 : La conception de circuits quantiques capables d'exclure des états spécifiques est un défi technique important en informatique quantique, on parle de **state preparation**. Les circuits quantiques utilisées ici sont similaires à celles de [9], mais alors que [9] traite la censure de configurations complexes, ce travail se limite à l'exclusion d'un seul état, avec une distribution uniforme sur l'espace admissible. Cette approche simplifiée permet une mise en oeuvre efficace et reste compatible avec l'encodage binaire minimal.

Il existe donc des architectures efficaces pour censurer exactement un seul état, mais pour des cas plus complexes, on doit recourir à des techniques d'approximation. **Dans la suite, on fait l'hypothèse qu'une telle censure est toujours réalisable, afin de clarifier l'exposé de la méthode théorique proposée.**

Décomposition de C_5^* et identification avec $C_3 \times C_2^*$

Pour illustrer concrètement la construction d'un isomorphisme opératoire entre représentations particulières de l'espace binaire restreint C_5^* (avec partition optimale $I = (3, 2)$) et du produit \$C_3 $\times C_2^*$, on explicite ici la structure de l'isomorphisme entre C_5^* et \$C_3 $\times C_2^*$.

On rappelle que C_5^* est l'ensemble des mots binaires de longueur 5 dont la partition $I = (3, 2)$ donne :

- une première composante $q^{(1)} \in \{0, 1\}^3$ (3 bits),
- une seconde composante $q^{(2)} \in \{0, 1\}^2$ (2 bits), restreinte à $(0, 1), (1, 1), (1, 0)$ (on exclut $(0, 0)$).

On a donc $|C_5^*| = 8 \times 3 = 24$ éléments.

Générateurs et actions opératoires

Sur C_2^*

- Générateurs $\Sigma' = \{\gamma_1, \gamma_2\}$
- L'espace d'états $E' = \{(0, 1), (1, 1), (1, 0)\}$.
- L'action opératoire π' des γ_i est l'invertion du bit i.

Sur C_3

- Générateurs $\Sigma = \{\sigma_1, \sigma_2, \sigma_3\}$, où
- L'espace d'états $E = \{0,1\}^3$.
- L'action opératoire π est pour chaque σ_i invertion du i-ème bit.

la procédure opératoire sur C_3 est donnée par :

$$\mathcal{W}_{C_3} = (\sigma_2,\, \sigma_3,\, \sigma_1,\, \sigma_3,\, \sigma_2,\, \sigma_3,\, \sigma_1,\, \sigma_3)$$

appliquée à l'état initial $e_0 = (0,0,0)$.

Sur C_5^*

- Générateurs $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \gamma_1, \gamma_2\}$: chaque générateur agit soit sur la composante $q^{(1)}$ (bits 1 à 3), soit sur $q^{(2)}$ (bits 4 et 5)
- L'espace d'états $E = C_5^*$.

Procédure opératoire couvrante non cyclique sur C_5^*

la procédure d'énumération $\mathcal{W}_{C_5^*}$ est construite directement comme la concaténation suivante :

$$\mathcal{W}_{C_5^*} = (\sigma_2,\, \sigma_3,\, \sigma_1,\, \sigma_3,\, \sigma_2,\, \sigma_3,\, \sigma_1,\, \sigma_3,\, \gamma_1,\, \sigma_3,\, \sigma_1,\, \sigma_3,\, \sigma_2,\, \sigma_3,\, \sigma_1,\, \sigma_3,\, \gamma_2,\, \sigma_3,\, \sigma_1,\, \sigma_3,\, \sigma_2,\, \sigma_3,\, \sigma_1,\, \sigma_3) \tag{10}$$

Isomorphisme partiel de représentation opératoire explicite entre C_5^* et $C_3 \times C_2^*$

On définit alors l'isomorphisme d'énumération :

- **Application φ sur les générateurs** : chaque générateur de C_5^* (invertion d'un bit) est identifié à un générateur correspondant de C_3 (pour les 3 premiers bits) ou de C_2^* (pour les 2 derniers bits), c'est-à-dire

$$\varphi(\text{invertion du bit } i) = \begin{cases} \sigma_i & \text{si } i \in \{1,2,3\} \\ \gamma_{i-3} & \text{si } i \in \{4,5\} \end{cases}$$

- **Application θ sur les états** : chaque mot $q = (q^{(1)}, q^{(2)}) \in C_5^*$ est envoyé sur le couple $(q^{(1)}, q^{(2)}) \in C_3 \times C_2^*$.

Table 4: Tableau de correspondance opératoire entre C_5^* et $C_3 \times C_2^*$

Étape	$\mathcal{W}_{C_5^*}$	$q^{(1)}$ **(3 bits)**	$q^{(2)}$ **(2 bits)**	$(q^{(1)}, q^{(2)})$
0	–	(0,0,0)	(0,1)	(0,0,0,0,1)
1	σ_2	(0,1,0)	(0,1)	(0,1,0,0,1)
2	σ_3	(0,1,1)	(0,1)	(0,1,1,0,1)
3	σ_1	(1,1,1)	(0,1)	(1,1,1,0,1)
4	σ_3	(1,1,0)	(0,1)	(1,1,0,0,1)
5	σ_2	(1,0,0)	(0,1)	(1,0,0,0,1)
6	σ_3	(1,0,1)	(0,1)	(1,0,1,0,1)
7	σ_1	(0,0,1)	(0,1)	(0,0,1,0,1)
8	σ_3	(0,0,0)	(0,1)	(0,0,0,0,1)
9	γ_1	(0,0,0)	(1,1)	(0,0,0,1,1)
10	σ_3	(0,0,1)	(1,1)	(0,0,1,1,1)
11	σ_1	(1,0,1)	(1,1)	(1,0,1,1,1)
12	σ_3	(1,0,0)	(1,1)	(1,0,0,1,1)
13	σ_2	(1,1,0)	(1,1)	(1,1,0,1,1)
14	σ_3	(1,1,1)	(1,1)	(1,1,1,1,1)
15	σ_1	(0,1,1)	(1,1)	(0,1,1,1,1)
16	σ_3	(0,1,0)	(1,1)	(0,1,0,1,1)
17	γ_2	(0,1,0)	(1,0)	(0,1,0,1,0)
18	σ_3	(0,1,1)	(1,0)	(0,1,1,1,0)
19	σ_1	(1,1,1)	(1,0)	(1,1,1,1,0)
20	σ_3	(1,1,0)	(1,0)	(1,1,0,1,0)
21	σ_2	(1,0,0)	(1,0)	(1,0,0,1,0)
22	σ_3	(1,0,1)	(1,0)	(1,0,1,1,0)
23	σ_1	(0,0,1)	(1,0)	(0,0,1,1,0)
24	σ_3	(0,0,0)	(1,0)	(0,0,0,1,0)

On observe sur la Table 4 que la procédure d'énumération ainsi construite permet de parcourir exhaustivement les 24 états distincts de C_5^* via la procédure opératoire décrite. Il est important de souligner a nouveau que **cette trajectoire n'est pas cyclique** : la trajectoire initiée depuis l'état $(0, 0, 0, 0, 1)$ ne s'achève pas sur ce même état, bien que ce sommet soit effectivement visité une et une seule fois (précisément à l'étape 8).

2.2.4 Construction d'un Isomorphisme d'énumération entre $C_{n'}^*$ et S_n

Cette construction s'étend de manière naturelle à tout espace binaire restreint $C_{n'}^*$. Il suffit, pour cela, de déterminer une partition optimale I de la dimension n', puis de définir, sur chaque bloc issu de cette partition, une représentation opératoire adaptée. La procédure globale d'énumération s'obtient alors par concaténation séquentielle des procédures opératoires associés à chaque bloc, conformément à la structure imposée par I.

Construction algorithmique générale pour l'énumération par blocs alternant

Étape 1 : Détermination de la partition optimale

Soit N le nombre d'éléments à représenter (par exemple, $N = n!$ pour S_n). On cherche une partition $I = (a_1, a_2, \dots, a_k)$ de la dimension totale $n' = \lceil \log_2(N) \rceil$ telle que :

- $N = \prod_{j=1}^{k} M_j$, où chaque M_j est le nombre d'états retenus dans le bloc j (typiquement $M_j = 2^{a_j}$ ou $2^{a_j} - m_j$ si certains états sont interdits).
- On minimise le nombre de tuples interdits dans chaque bloc, en choisissant a_j pour que M_j soit le plus proche possible d'un facteur de N.

Étape 2 : Construction des blocs binaires

Pour chaque bloc j :

- On définit l'espace E_j des états binaires de taille a_j.
- On exclut explicitement les m_j tuples interdits pour obtenir M_j états valides.
- On construit une procédure opératoire couvrante sur E_j (par exemple, un code de Gray de longueur M_j).

Étape 3 : Alternance et concaténation

La procédure opératoire globale $\mathcal{W}$ sur $C_{n'}^*$ est obtenue par alternance des procédures sur chaque bloc, selon le principe suivant :

- On parcourt le premier bloc E_1 selon sa procédure opératoire (par exemple, code de Gray sur M_1 états).

- À chaque fois que le parcours du bloc E_1 est terminé (i.e., après M_1 étapes), on incrémente le bloc E_2 selon sa procédure opératoire, et on recommence le parcours de E_1.
- Ce schéma se répète pour chaque bloc : le bloc E_j est incrémenté à chaque fois que le parcours complet du bloc précédent E_{j-1} est achevé, selon sa propre procédure opératoire.

Étape 4 : Procédure de ranking

La procédure de ranking sur l'espace binaire partitionné repose sur une logique arithmétique fondée sur la factorisation de l'espace des états selon une partition optimale I comme précédemment définie. Soit $s \in \{0,1\}^{n'}$ un mot binaire.

On procède comme suit :

1. **Découpage du mot binaire** : s est scindé en sous-mots selon la partition I, chaque bloc s_j correspondant à un bloc de bits de taille a_j.
2. **Calcul du rang local** : Pour chaque bloc j, on associe à s_j son rang local r_j dans l'ordre des configurations admissibles du bloc, typiquement déterminé par la procédure de Gray ou toute autre énumération couvrante sur le sous-espace correspondant.
3. **Agrégation pondérée des rangs** : Le rang R du mot s est reconstruit par une somme pondérée des rangs locaux, chaque rang étant multiplié par un facteur de pondération (modulo) qui encode la cardinalité des blocs déjà traités. Formellement, pour une suite de blocs $(s_1, \dots, s_k)$, on pose :

$$R = \sum_{j=1}^{k} r_j \cdot \left(\prod_{l=1}^{j-1} M_l \right) \tag{11}$$

où le produit est pris sur les tailles des blocs précédents, assurant ainsi la compatibilité lexicographique de l'indexation.

Exemple explicite pour 3 blocs de taille 3

Pour illustrer ce mécanisme, on considère sur la Figure 6 l'exemple d'une concaténation sur trois blocs de taille 3 (soit $3 \times 3 \times 3 = 27$ éléments). Chaque bloc est représenté par une couleur distincte sur la figure correspondante, mettant en évidence la dynamique de l'énumération.

Soit le mot $s = (1, 3, 1)$, où :

- 1 correspond au premier bloc (rouge),
- 3 au deuxième bloc (vert),
- 1 au troisième bloc (bleu).

Pour calculer le rang global R de ce mot dans l'énumération par blocs, on applique la formule de ranking :

$$R = r_1 + r_2 \times M_1 + r_3 \times (M_1 \times M_2) \tag{12}$$

où :

- $M_1 = 3$ (taille du bloc rouge), $M_2 = 3$ (taille du bloc vert).
- $r_1 = 0$ (pour 1 rouge),
- $r_2 = 2$ (pour 3 vert),
- $r_3 = 0$ (pour 1 bleu).

On calcule alors :

$$R = 0 + 2 \times 3 + 0 \times (3 \times 3) = 0 + 6 + 0 = 6$$

Interprétation : Le mot 131 (1rouge, 3vert, 1bleu) correspond donc au rang 6 (en commençant a 0) dans l'énumération globale des 27 éléments.

Ce calcul montre comment la structure par blocs permet d'indexer efficacement chaque mot binaire dans l'ordre de l'énumération, en utilisant la pondération des rangs locaux par la taille des blocs précédents.

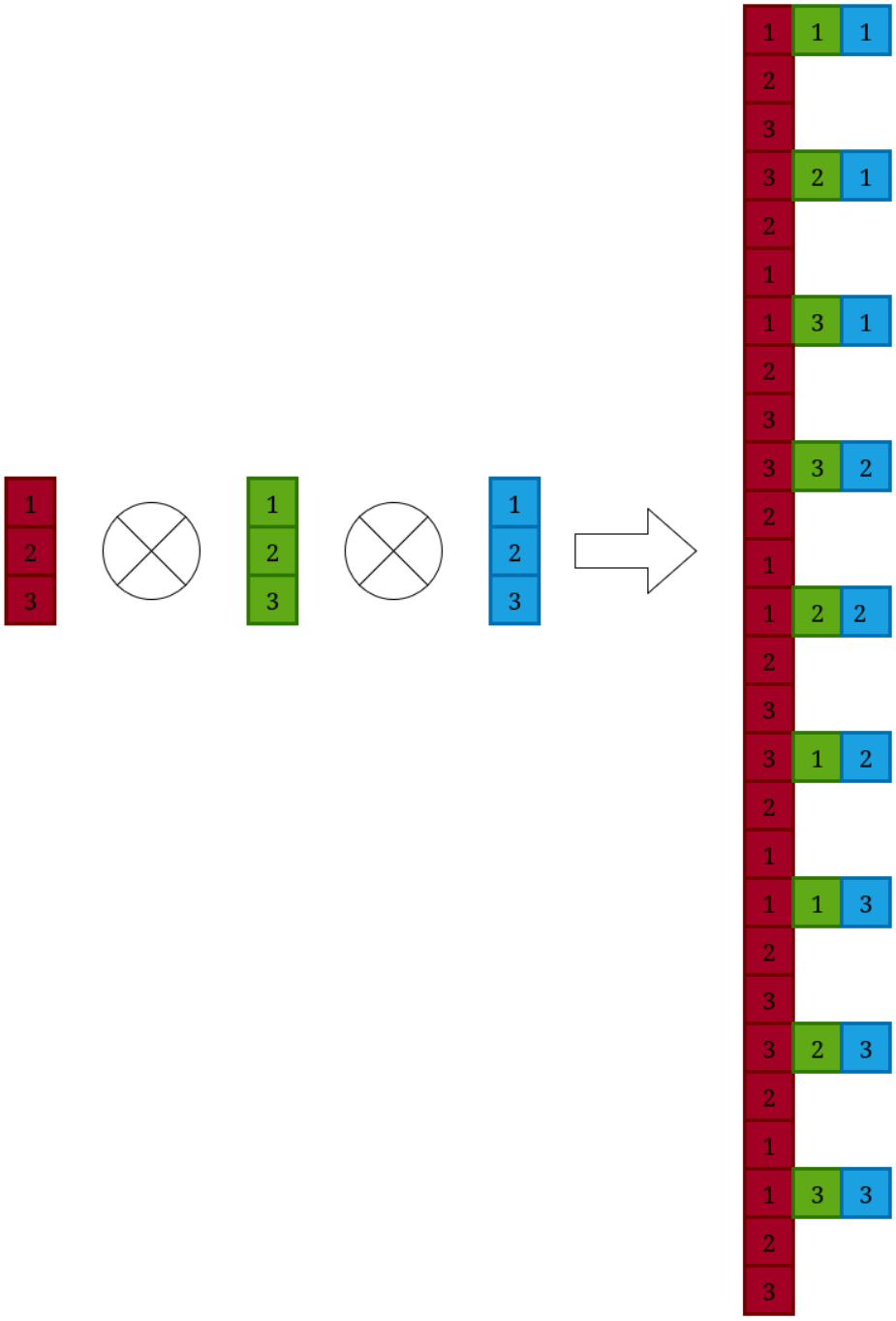

Figure 6: Exemple de construction d'une énumération par blocs

De façon générale, cette méthode permet de construire explicitement une énumération couvrante sur $C^*_{n'}$, en exploitant la factorisation induite par la partition I.

Exemple : $n = 8$**, partition** $I = (7, 2, 4, 3)$

On considère ici le cas $n = 8$, pour lequel $8! = 40320$ états doivent être représentés. La partition optimale choisie est $I = (7, 2, 4, 3)$, correspondant à une décomposition $128 \times 3 \times 15 \times 7 = 40320$.

- **Bloc 1** : 7 bits, $2^7 = 128$ états
- **Bloc 2** : 2 bits, $2^2 = 4$ états, mais on n'en garde que 3 (par exemple, on exclut $(0, 0)$)
- **Bloc 3** : 4 bits, $2^4 = 16$ états, mais on n'en garde que 15 (par exemple, on exclut $(0, 0, 0, 0)$)
- **Bloc 4** : 3 bits, $2^3 = 8$ états, mais on n'en garde que 7 (par exemple, on exclut $(0, 0, 0)$)

Procédure opératoire

Pour construire une procédure opératoire couvrante sur C^*_{16}, on procède par concaténation des procédures sur chaque bloc :

- Sur le bloc $q^{(1)}$ (7 bits) : Code de Gray de longueur 128.
- Sur le bloc $q^{(2)}$ (2 bits, 3 états) : Code de Gray de longueur 3 avec (0,0) interdit.
- Sur le bloc $q^{(3)}$ (4 bits, 15 états) : Code de Gray de longueur 15 avec (0,0,0,0) interdit.
- Sur le bloc $q^{(4)}$ (3 bits, 7 états) : Code de Gray de longueur 7 avec (0,0,0) interdit.

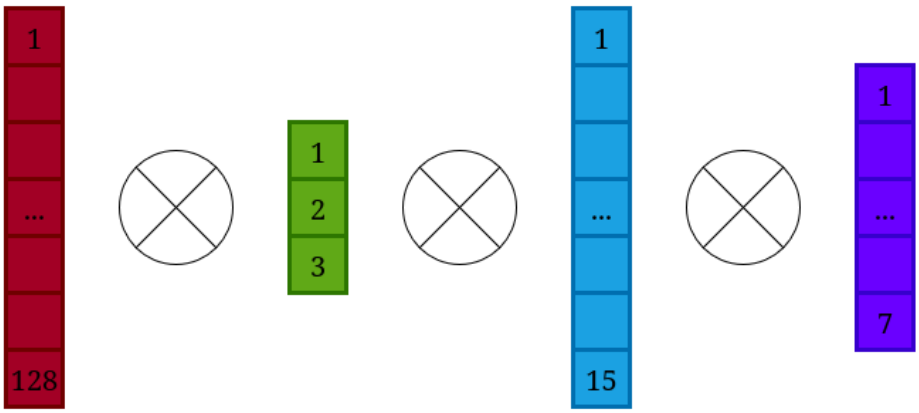

Figure 7: Schéma de la concaténation par blocs selon la partition optimale

2.3 : Généralisations et extensions

2.3.1 Généralisation à toutes représentations régulières d'un groupe.

La méthode systématique développée pour construire et parcourir C_n^* aboutit à un chemin hamiltonien (et non plus un cycle), ce qui modifie la structure de l'énumération. Pour S_n, il devient alors naturel de chercher une procédure analogue, adaptée à cette nouvelle structuration. On propose ici une première généralisation : puisque le code de Gray a été découpé en blocs, il est pertinent d'envisager une découpe similaire pour S_n.

Concrètement, cela consiste à partitionner S_n en orbites sous l'action d'un de ses sous-groupe $H \subset S_n$. Par exemple, pour S_8, il existe un sous-groupe de taille 384. On peut alors construire une procédure opératoire couvrante sur ce sous-groupe, puis la répliquer sur chacune des 105 orbites (puisque $8! = 384 \times 105$), en choisissant à chaque fois un point de départ distinct.

Remarque 10: : Le très célèbre **théorème de Cayley** [10] établis que tout groupe peut être plongé dans un sous-groupe d'un groupe symétrique. Un tel plongement donne lieu à une représentation classique du groupe sous forme de permutations ; on parle alors de représentation régulière. Cette méthode permet d'exploiter tout ce que l'on connaît des représentations régulières pour l'appliquer aux représentations opératoires.

2.3.2 Partitions et énumérations par blocs.

Il existe plusieurs décompositions possibles de S_8 en produits d'orbites par l'action de l'un de ces sous-groupes. On cite en Table 5 les partitions testées expérimentalement pour différents sous-groups de S_8.

On formalise la construction d'une énumération par blocs de l'espace d'état $E = \{(i_1, \dots, i_8) \mid (i_1, \dots, i_8)$ permutation de $(1, \dots, 8)\}$, afin d'illustrer la méthode générale exposée précédemment.

Cette approche consiste à partitionner l'ensemble des $8! = 40320$ permutations en un tableau où :

- chaque ligne correspond à la trajectoire opératoire issue de l'application de la procédure opératoire sur le sous-groupe considéré ;
- chaque colonne représente une classe d'équivalence d'instances pour l'action du sous-groupe considéré.

On se concentre ici sur la décomposition $8! = 384 \times 105$, qui correspond à une partition binaire $(7, 2) \mid (3, 4)$.

Table 5: Décompositions en sous-group et partitions binaires pour S_8

Taille du groupe	Décomposition de 8!	Partition binaire (trajectoires | départ)	Facteurs premiers utilisés
64	$8! = 64 \times 630$	$(6) \mid (1,2,3,4)$	$64 = 2^6, \quad 630 = 2 \times 3 \times 5 \times 7$
128	$8! = 128 \times 315$	$(7) \mid (2,3,4)$	$128 = 2^7, \quad 315 = 3^2 \times 5 \times 7$
192	$8! = 192 \times 210$	$(6,2) \mid (1,3,4)$	$192 = 2^6 \times 3, \quad 210 = 2 \times 3 \times 5 \times 7$
384	$8! = 384 \times 105$	$(7,2) \mid (3,4)$	$384 = 2^7 \times 3, \quad 105 = 3 \times 5 \times 7$
720	$8! = 720 \times 56$	$(4,4,2) \mid (3,3)$	$720 = 2^4 \times 3^2 \times 5, \quad 56 = 2^3 \times 7$

Le sous-groupe de taille 384 utilisé, noté G_{384}, est engendré par les permutations suivantes (écrites en notation cyclique) :

- $(1\,4)(2\,3)$
- $(1\,2)$
- $(5\,8)(6\,7)$
- $(5\,6)$
- $(3\,6)(4\,5)$
- $(3\,4)$

L'action de ces générateurs sur l'espace des états E est donnée par la composition matricielle, comme dans les exemples précédents.

La procédure opératoire couvrante sur G_{384} est construite explicitement à partir de ces générateurs ; la séquence complète est fournie en annexe.

Pour obtenir une énumération complète de E, on sélectionne une liste de 105 représentants e_0 dans E, chacun correspondant à un point de départ distinct pour l'orbite sous l'action de G_{384}. La liste exhaustive de ces représentants est également donnée en annexe.

Du côté binaire, la représentation de Gray utilisée est celle décrite dans la section précédente et en Table 5 : les deux premiers blocs binaires (de tailles 7 et 2) codent la trajectoire du groupe G_{384} , tandis que les deux autres blocs (de tailles 3 et 4) servent à indexer le choix du point de départ parmi les 105 orbites.

Afin d'illustrer l'impact structurel de la construction opératoire décrite ci-dessus, les Figure 8 et Figure 9 présentent une analyse comparative du paysage des coûts pour **une instance aléatoire du problème du voyageur de commerce de taille 8.**

Plus précisément, chaque solution est positionnée dans une matrice dont **les lignes correspondent aux trajectoires générées par l'action du sous-groupe de taille 384, et les colonnes aux différents points de départ (représentants d'orbites).**

Le coût de chaque tournée, calculé comme la somme des distances entre les points visités (Comme dans l'éxemple en (1) en début de la partie 2), est placé à la position correspondante dans le tableau.

Cette organisation matricielle n'est pas arbitraire : elle découle directement de la structuration mise en place, Grâce à l'isomorphisme opératoire, l'algorithme maître M accède aux tournées en tirant parti de cette organisation : les labels binaires générés par le circuit quantique sont structurés en blocs, ce qui fait que les lignes et colonnes adjacentes dans la matrice correspondent à des labels proches au sein de chaque bloc.

Cette propriété est essentielle pour optimiser la recherche, car elle garantit que les modifications locales dans les labels binaires se traduisent par des variations contrôlées dans les tournées générées, facilitant ainsi l'exploration et l'exploitation des régions de faible coût dans le paysage du TSP.

L'analyse visuelle révèle de manière manifeste l'apparition d'une organisation en grille induite par la construction opératoire : les colonnes regroupent des ensembles de solutions présentant des coûts similaires (faibles ou élevés), traduisant une forte corrélation structurelle entre les points de départ et les trajectoires opératoires. Cette structuration, absente dans le cas d'une attribution aléatoire, met en évidence le potentiel de la méthode proposée.

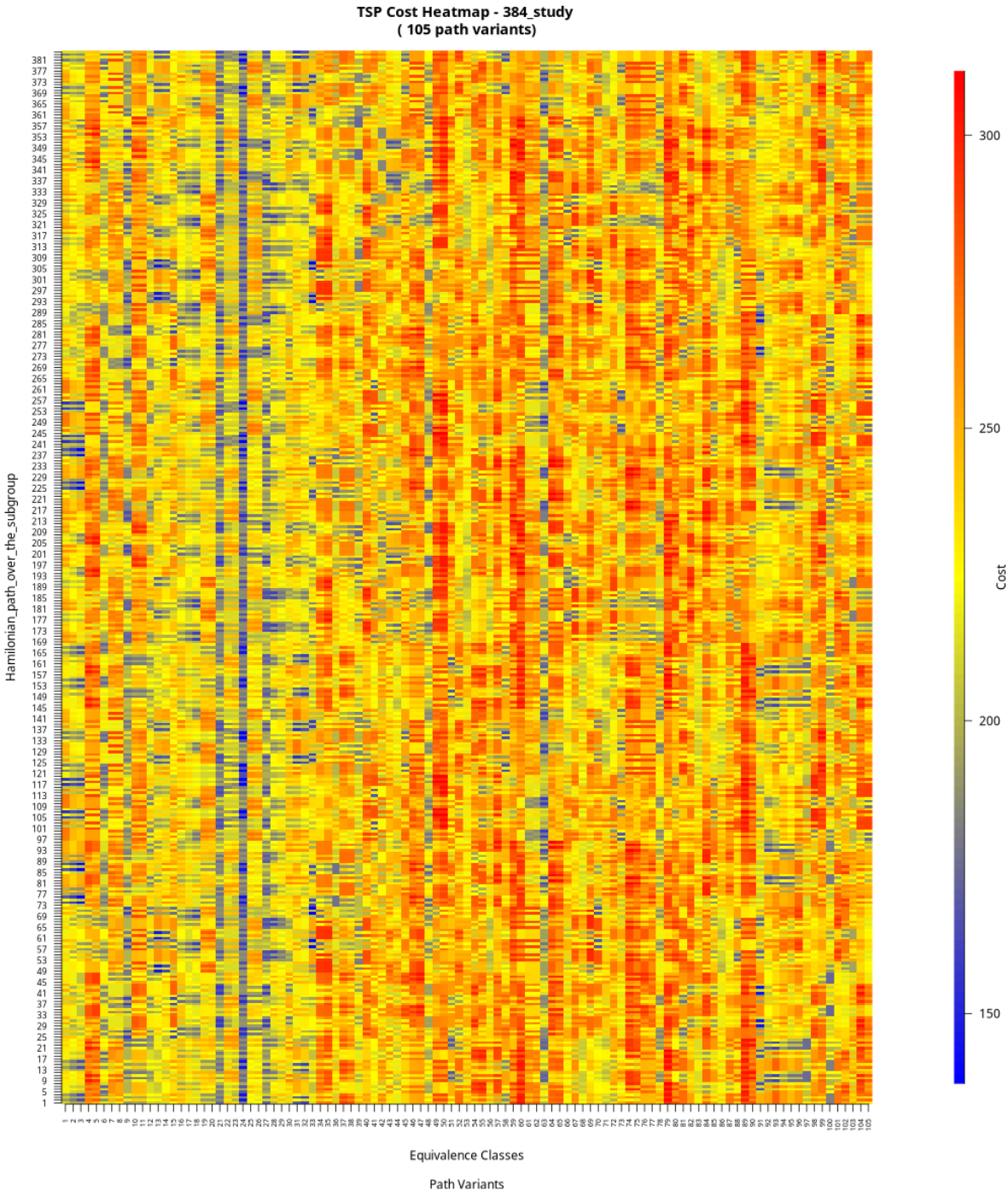

Figure 8: Organisation structurée des coûts issue de la représentation opératoire de G_{384}

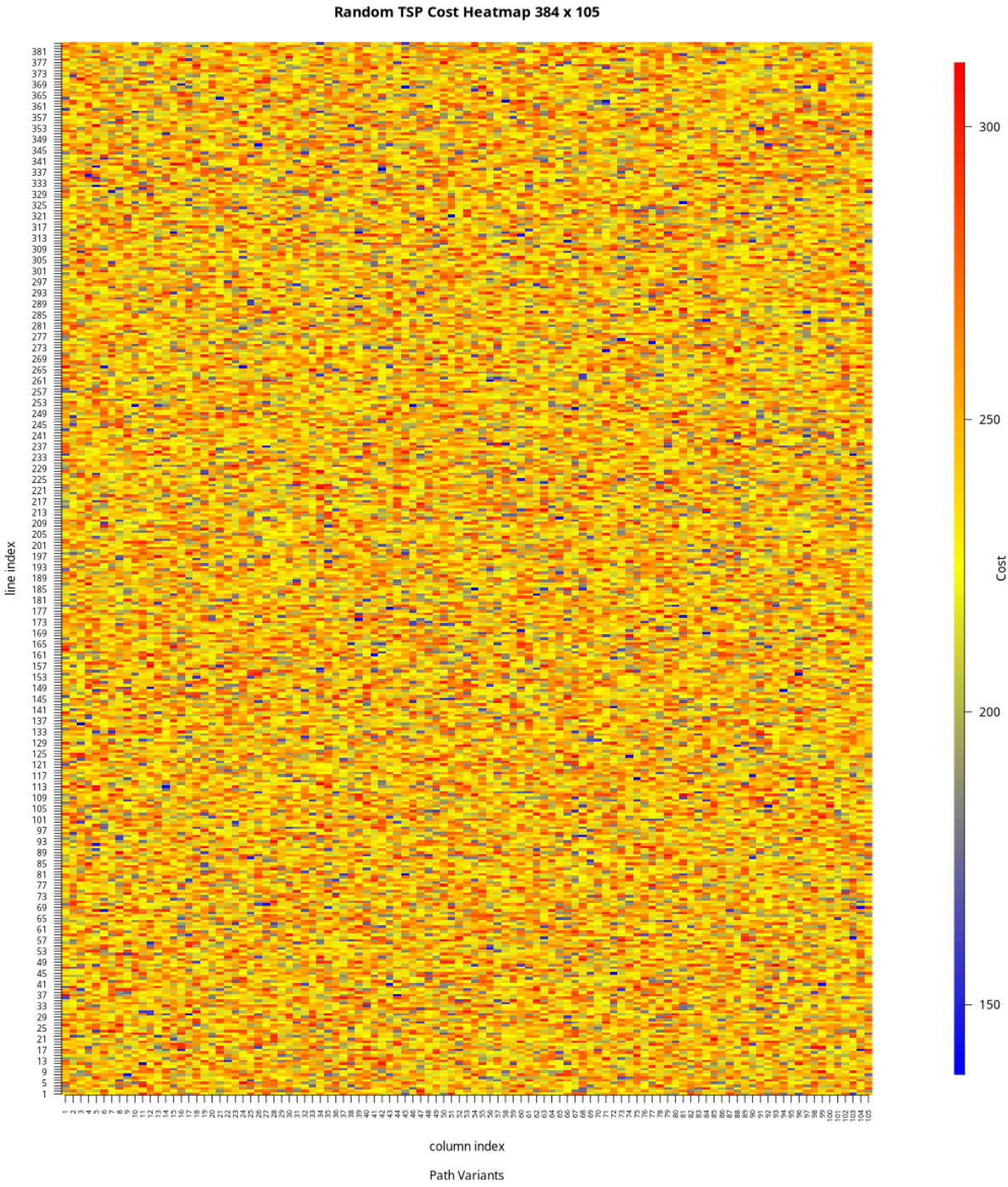

Figure 9: Répartition aléatoire des coûts au même format que G_{384} pour comparaison

2.3.3 Entre énumérations et labellisations.

On peut désormais apprécier tout l'intérêt du formalisme introduit : il permet d'analyser finement la dynamique opératoire induite par un sous-groupe donné, en étudiant l'impact de ses trajectoires à partir de différents points de départ dans l'espace d'états E.

Cette approche offre la possibilité de choisir explicitement le niveau de structuration de E : plus le sous-groupe considéré est de grande taille, plus ses trajectoires couvrent une portion importante de E, et moins il subsiste de points de départ distincts pour les orbites. On dispose ainsi d'un outil flexible pour organiser E selon des critères algébriques ou combinatoires, et pour faire correspondre cette structuration à celle des états quantiques.

En particulier, en organisant les blocs binaires selon la décomposition en sous-groupes et en points de départ, on permet à l'algorithme maître M de distinguer, au sein de la représentation binaire, les bits associés à la dynamique d'énumération (c'est-à-dire, ceux qui sont effectivement en isomorphisme opératoire partielle pour la procédure du sous-groupe considéré) de ceux qui relèvent de la labellisation des points de départ (et donc, d'une structure plus faible ou arbitraire).

On précise que **tous les détails complémentaires relatifs aux constructions opératoires sont disponibles en annexe** : on y retrouve la liste exhaustive des générateurs utilisés pour chaque sous-groupe testé, la description des groupes considérés, les cycles hamiltoniens explicitement trouvés sur leurs graphes de Cayley, ainsi que des visualisations graphiques de ces graphes de Cayley (représentations des trajectoires opératoires, organisation des orbites, etc.). Ces ressources permettent d'approfondir la compréhension des méthodes employées et d'analyser en détail la structure des procédures opératoirespour chaque cas étudié.

De même, les résultats détaillés des différentes implémentations sont également présentés en annexe pour l'ensemble des groupes et générateurs testés. Dans la Partie 3, seuls les résultats correspondant à la meilleure configuration groupe/générateurs identifiée sont exposés dans le corps du texte, afin d'en faciliter la lecture et la clarté. Toutefois, de nombreux autres groupes et variantes de générateurs ont été explorés expérimentalement ; leurs performances et analyses comparatives sont laissées en annexe pour permettre au lecteur d'approfondir la compréhension des choix effectués et d'apprécier la diversité des configurations étudiées.

Partie 3 : Implémentations et résultats

Initialement, l'algorithme maître M utilisé était un algorithme de gradient appelé F-VQE. Les détails de son fonctionnement importent peu ici, puisque, comme rappelé précédemment, toute la construction se veut agnostique vis-à-vis du choix de M. Ce qui importe, c'est que l'implémentation initialement réaliser (noté **Base_gray_trotter_method** dans les graphiques qui suivront) se résume ainsi de la manière suivante :

- On construit un circuit quantique comportant $log_2(n!) + 1$ qubits.
- On prépare le ranking d'un code de Gray associé aux $2^{log_2(n!)+1}$ états possibles.
- On prépare également l'unranking selon le schéma de Trotter-Johnson, associé aux $n!$ solutions.
- Si le ranking d'un mot binaire dépasse le $n!$-ième rang, on y associe l'unranking de ce rang modulo $n!$.

Cette procédure garantit que chaque état est toujours associé à une solution. Toutefois, elle présente plusieurs inconvénients déjà discutés en Partie 1 : certaines solutions sont surreprésentées dans le circuit, et bien que deux solutions adjacentes (au sens de Trotter-Johnson) partagent au moins un label adjacent (au sens de Gray) , cette structure est, minimale et peu expressive en pratique.

Cela dit, cette première méthode proposée pour F-VQE a déjà permis d'obtenir des résultats très prometteurs : à notre connaissance, aucun encodage aussi compact — c'est-à-dire minimal en nombre de qubits tout en représentant l'ensemble des solutions au moins une fois — n'avait encore été proposé. De plus, les portes utilisées restent très limitées, avec une complexité linéaire en le nombre de qubits. La Figure 10 illustre le potentiel de la méthode d'encodage en $\log_2(n!)$: il s'agit d'une tournée optimale sur 16 villes françaises, obtenue avec la méthode de base **Base_gray_trotter_method**

L'ensemble de ces expériences est réalisé sur simulateur, codé en Python via la librairie Qiskit.

Figure 10: Illustration du meilleur résultat de la méthode d'encodage en $log_2(n!)$ après simulation sur émulateurs quantiques.[11]

3.1 : Modifications de l'algorithme initial

Les modifications apportées à l'algorithme initial reposent sur la construction théorique de la Partie 2.

Le design du circuit quantique a été adapté pour permettre la censure explicite d'un états particuliers (l'état $(0, 0, 0, ..., 0)$). Cette capacité à couper certains états rend possible l'utilisation directe de la construction binaire restreinte $C^*_{n'}$, de sorte que le nombre de labels binaires présents dans le circuit correspond exactement au nombre de solutions du problème (par exemple, $n!$ pour S_n). On n'a donc plus besoin de sur-représenter certaines solutions ou d'introduire des labels supplémentaires : chaque état du circuit est associé de façon unique à une solution valide, ce qui optimise l'utilisation des ressources quantiques et garantit un encodage parfaitement compact.

Dans le cas où le problème du TSP est de taille $n > 11$, il devient nécessaire, comme on l'a vu, de concevoir un circuit quantique capable d'exclure précisément 11 états sur 16, ce qui s'avère particulièrement complexe à réaliser en pratique. Pour pallier cette difficulté, une stratégie alternative est adoptée : le circuit de l'ancienne méthode, basé sur une architecture de type **p-layer-ansatz** (un design de circuit très simple, disponible en annexe), est conservé, mais l'interprétation des états mesurés s'effectue par blocs, comme si le circuit avait été effectivement construit selon une partition en blocs.

Concrètement, lors de la lecture d'un mot binaire issu de la mesure, si l'un des blocs correspond à une configuration interdite par la procédure d'énumération, on associe à cet état le rang valide le plus proche dans la structure opératoire. Cette méthode permet d'approcher localement la dynamique opératoire souhaitée, même si le circuit quantique ne réalise pas explicitement la censure des états interdits.

Dans la suite, on distinguera donc ces deux variantes principales de la nouvelle méthode :

- **p_layer_circuit** : il s'agit de la version où le circuit quantique est construit selon une architecture p_layer_ansatz classique, sans censure explicite des états interdits. L'interprétation des états mesurés s'effectue par blocs, en associant à chaque configuration interdite le rang valide le plus proche dans la structure opératoire. Cette approche permet d'approximer la dynamique opératoire souhaitée, même si le circuit ne réalise pas explicitement la censure.
- **censored_circuit** : il s'agit de la version idéale, dans laquelle le circuit quantique est effectivement construit pour censurer explicitement les états interdits, conformément à la partition optimale et à la structure de $C^*_{n'}$. Chaque état mesuré correspond alors de façon unique à une solution valide, sans approximation ni sur-représentation.

Ces deux variantes seront utilisées dans les expérimentations qui suivent, afin de comparer l'impact de la censure explicite des états sur la performance de l'algorithme hybride.

3.2 Implémentation sur S_4

À chaque itération de l'algorithme F-VQE, le protocole expérimental s'appuie sur le fonctionnement précis du circuit quantique et sur la dynamique de l'optimisation hybride. Le circuit, construit selon la méthode d'encodage opératoire décrite précédemment (Partie 2), prépare un état quantique superposé sur l'espace binaire restreint $C^*_{n'}$, garantissant que chaque configuration mesurée correspond à une solution valide du problème.

Lors de la phase de mesure (étape 2 de la Figure 1), chaque tirage du circuit quantique produit un mot binaire, qui est décodé via la procédure d'unranking opératoire pour reconstituer la solution combinatoire associée. Pour chaque configuration mesurée, on calcule le coût c_i correspondant à la solution reconstruite. Le nombre d'occurrences n_i de chaque configuration au cours des N_{shots} mesures reflète la probabilité de l'état quantique associé, telle qu'elle résulte de la distribution des états induite par les paramètres du circuit à cette itération.

Le coût moyen C pour l'itération courante est alors défini par :

$$C = \frac{\sum_i c_i \cdot n_i}{N_{\text{shots}}} \tag{13}$$

Cette moyenne pondérée intègre la structure probabiliste du circuit : les solutions les plus probables (i.e., les états quantiques les plus amplifiés par l'algorithme) contribuent davantage au coût moyen, traduisant l'efficacité de l'optimisation paramétrique du circuit.

Ce processus est répété à chaque étape de l'algorithme F-VQE, où les paramètres du circuit sont mis à jour par l'algorithme (étape 1 de la Figure 1), en fonction du coût moyen observé. La dynamique itérative vise à minimiser C en adaptant la distribution quantique pour amplifier les solutions de faible coût.

Pour évaluer la performance de l'algorithme, on calcule à chaque itération l'écart relatif à la solution optimale connue C_{opt} :

$$\text{gap} = \left(\frac{C}{C_{\text{opt}}} \times 100 \right) - 100 \tag{14}$$

Cet indicateur quantifie, en pourcentage, la déviation du coût moyen obtenu par le circuit par rapport à l'optimum. Un gap nul signifie que la distribution quantique est entièrement concentrée sur la solution optimale, tandis qu'un gap positif traduit la dispersion résiduelle sur des solutions sous-optimales, révélant la marge d'amélioration possible dans la calibration du circuit ou dans la structure opératoire choisie.

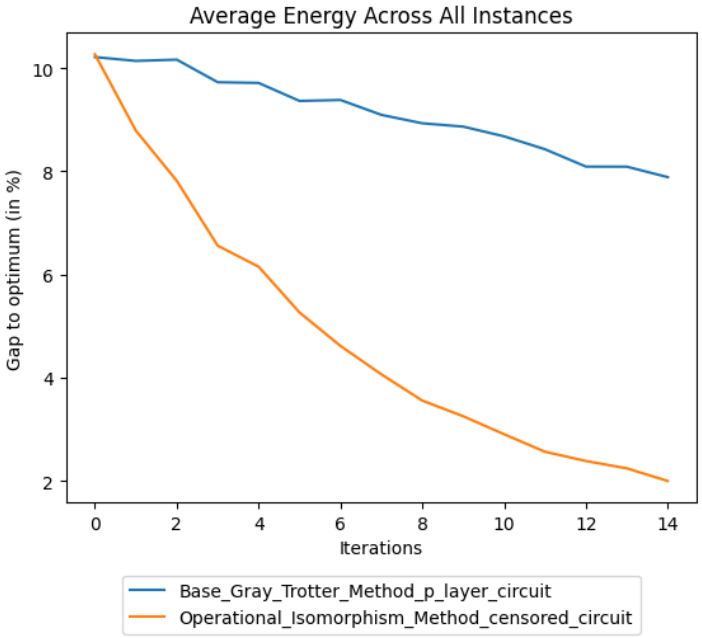

Figure 11: Évolution du coût moyen (gap) au fil des itérations pour S_4 (moyenne sur 100 instances)

La Figure 11 présente, en abscisse, les itérations de l'algorithme F-VQE, et en ordonnée, le gap moyen calculé sur les 100 instances générées.

On observe que la nouvelle méthode **(notée OIM : Operational Isomorphism Method)** permet de réduire significativement le gap moyen au fil des itérations, atteignant environ 2 % en fin d'optimisation, contre 8 % pour l'ancienne approche.

L'histogramme présenté en Figure 12 illustre la distribution du nombre d'instances pour lesquelles le gap obtenu est inférieur à 1 % de l'optimum, sur 100 instances testées. On constate que la nouvelle méthode permet d'atteindre ce seuil pour 38 instances sur 100, contre seulement 2 instances pour l'ancienne méthode. Cette amélioration significative témoigne de l'efficacité accrue de la construction opératoire proposée, qui favorise une meilleure concentration de la distribution quantique sur les solutions optimales.

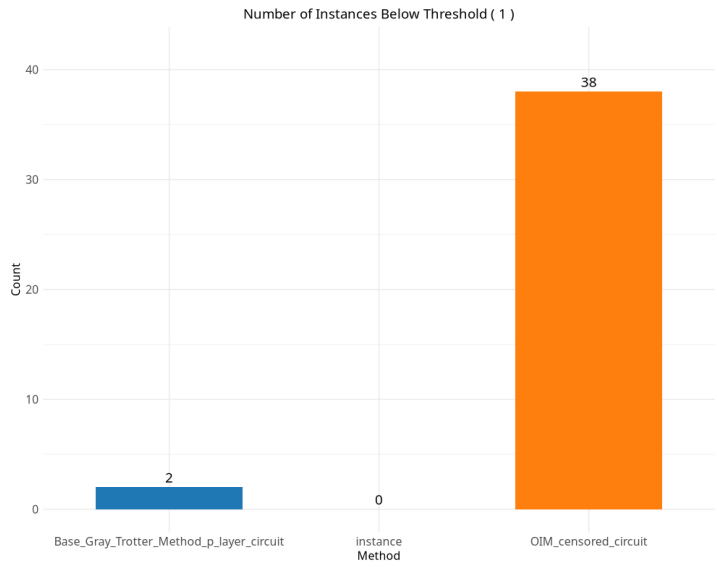

Figure 12: Histogramme du nombre d'instances atteignant un gap inférieur à 1 % de l'optimal

3.3 Implémentation sur S_8

F-VQE

Pour le cas de S_8, les résultats obtenus présentent des caractéristiques remarquablement distinctes de ceux observés pour S_4, et soulèvent des questions d'ordre structurel. Plusieurs sous-groupes de tailles variées ont été examinés expérimentalement (les courbes suivant le même procédé que Figure 13 sont disponibles en annexe pour les différents groupes testés) , révélant une tendance inattendue : la performance moyenne de l'algorithme s'améliore significativement lorsque la taille du sous-groupe considéré est proche de $\sqrt{n!}$. Cette amélioration est particulièrement marquée lorsque le nombre de shots est faible, c'est-à-dire lorsque l'estimation de la distribution de probabilité repose sur un échantillonnage limité.

De façon notable, il apparaît sur la Figure 13 que la variante **p_layer_circuit** de la nouvelle méthode offre les meilleures performances, pour le sous-groupe de taille 384 (celui ayant obtenu les résultats les plus probants à 50 shots). Autrement dit, l'approximation réalisée par le circuit complet, sans censure explicite des états interdits, surpasse la version du circuit idéal pour F-VQE. Ce constat est contre-intuitif, puisque l'on pourrait s'attendre à ce que la censure stricte des états non valides améliore systématiquement la qualité de l'optimisation. Plusieurs pistes explicatives ont été envisagées, notamment la manière spécifique dont F-VQE évalue la distribution des états du circuit, mais aucune explication définitive n'a pu être établie à ce jour.

On observe sur l'histogramme Figure 14 que l'ancienne méthode atteint le seuil de 1 % de gap pour 12 instances sur 40, tandis que la nouvelle méthode (p-layer circuit) l'atteint pour

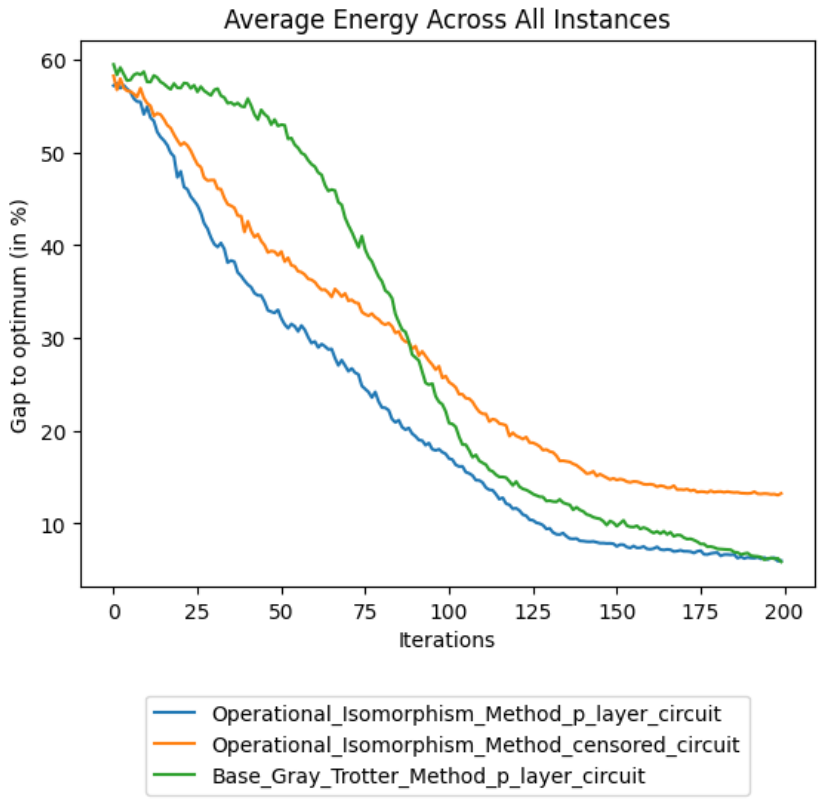

Figure 13: Évolution du gap moyen au fil des itérations pour S_8 (F-VQE, 40 instances)

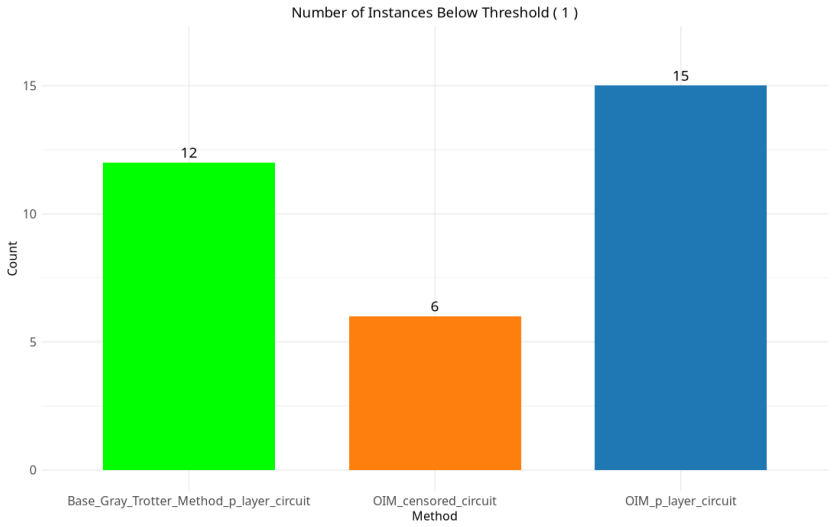

Figure 14: Histogramme du nombre d'instances atteignant un gap inférieur à 1 % de l'optimal(S_8, F-VQE, 40 instances)

15 instances sur 40. Les résultats sont donc globalement similaires, mais on note un léger avantage pour la nouvelle méthode. Ce constat est cohérent avec les temps moyens d'atteinte du seuil de 1 % reportés ci-dessous : la nouvelle méthode (**OIM_censored_circuit**) converge en moyenne plus rapidement (112 itérations) que l'ancienne (132 itérations), et la variante p-layer circuit converge en 119 itérations.

Lorsque le nombre de shots augmente, l'ancienne méthode tend à reprendre l'avantage et à fournir de meilleures performances. Des graphiques complémentaires, présentés en annexe, illustrent en détail l'influence du nombre de shots sur les résultats obtenus avec F-VQE. Toutefois, étant donné que l'objectif principal de ce travail n'est pas l'étude approfondie de l'algorithme maître, cette question n'est pas développée davantage dans le corps du texte.

COBYLA

Afin de déterminer si ce phénomène relève d'une propriété intrinsèque de la structure opératoire ou s'il est spécifique à l'algorithme maître F-VQE, des expérimentations complémentaires ont été menées. Le même sous-groupe de taille 384, ainsi que la méthode classique, ont été testés en association avec un autre algorithmes maîtres, permettant d'analyser la robustesse et la généralité de ces observations.

L'optimisation avec COBYLA, une méthode de descente sans gradient, est largement utilisée en optimisation quantique. Dans ce cas, COBYLA a nécessité d'augmenter le nombre de shots de 50 à 1000 pour commencer à converger, et les taux de convergence observés sont nettement moins bons que ceux obtenus avec F-VQE.

Cependant, il apparaît là encore une préférence très marquée pour la nouvelle méthode: sur une moyenne de 40 instances, c'est cette version qui se distingue largement. Ce constat confirme l'intérêt de la nouvelle construction, qui s'avère non seulement plus performante avec F-VQE, mais également bien mieux adaptée à des algorithmes d'optimisation alternatifs comme COBYLA, ouvrant ainsi la voie à une généralisation de la méthode à des architectures variées.

L'histogramme Figure 16 montre que pour atteindre le seuil de 30 % de gap avec COBYLA, toutes les méthodes testées obtiennent des résultats similaires en termes de nombre d'instances franchissant ce seuil. Cependant, si l'on considère le temps moyen nécessaire pour atteindre ce seuil, la méthode **OIM_p_layer_circuit** se distingue nettement : elle permet d'atteindre le seuil plus rapidement (10.6 itérations en moyenne), contre 15.6 pour **OIM_censored_circuit** et 17.6 pour la méthode classique. Cela indique que, bien que la performance globale soit comparable entre les approches, l'architecture **OIM_p_layer_circuit** offre un avantage en rapidité de convergence vers la zone de solutions acceptables.

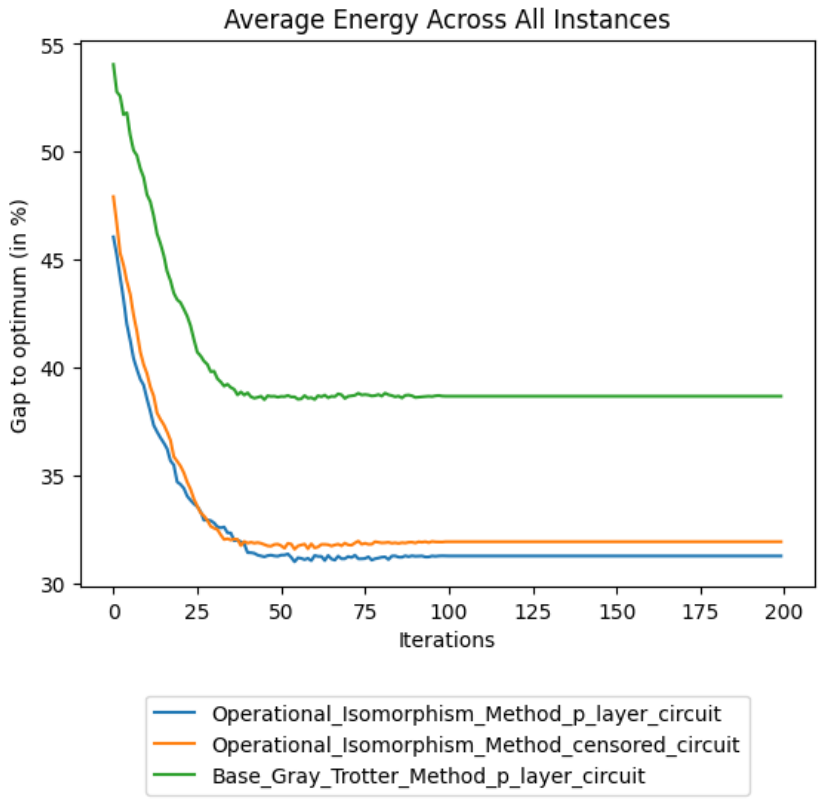

Figure 15: Évolution du gap moyen au fil des itérations pour S_8 (COBYLA, 40 instances)

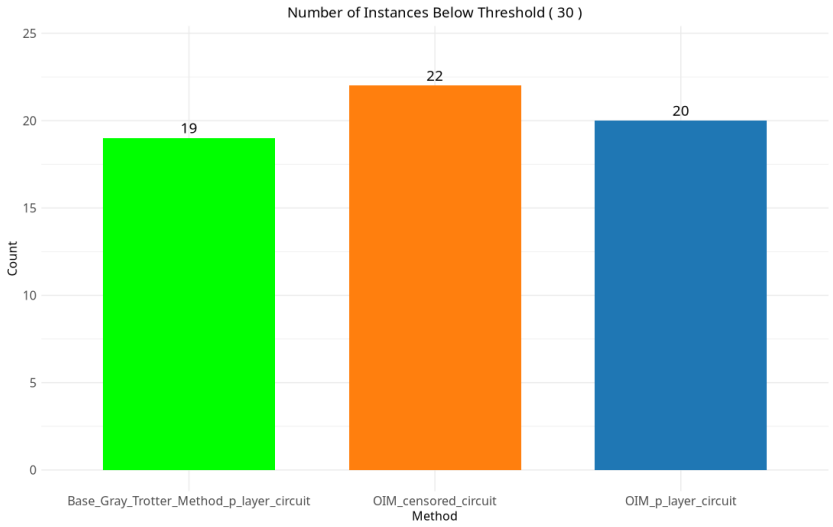

Figure 16: Histogramme du nombre d'instances atteignant un gap inférieur à 1 % de l'optimal (S_8, COBYLA, 40 instances)

3.4 Implémentation sur S_{16}

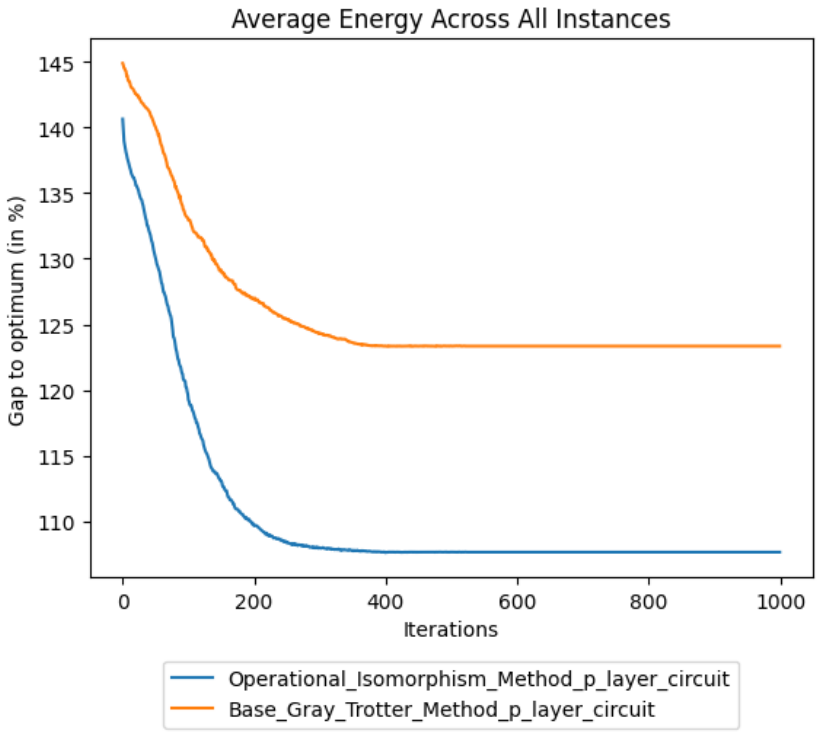

Figure 17: Évolution du gap moyen au fil des itérations pour S_{16} (COBYLA, 30 instances)

Dans le cas de S_{16}, les expérimentations (Figure 17) ont été menées exclusivement avec l'algorithme COBYLA, en raison du coût computationnel prohibitif associé à la simulation de F-VQE pour des espaces de cette taille. Il convient de souligner que les résultats obtenus avec COBYLA sont de pauvre qualité : avec un nombre de tirs limité à 10 000, l'algorithme parvient toujours à mieux performer avec la nouvelle méthode, mais reste très loin de l'optimal (avec plus de 100 % de gap à l'optimal pour les deux méthodes).

Pour le cas de S_{16}, le principal défi rencontré n'est plus lié à l'implémentation quantique ou à la gestion des qubits, mais à la construction explicite des procédures opératoires couvrantes sur des groupes de taille aussi considérable ($16! \approx 2 \times 10^{13}$ états). En effet, la grande limite de la nouvelle méthode réside dans la détermination préalable d'une trajectoire opératoire exhaustive (parcours hamiltonien ou chemin couvrant) sur le graphe de Cayley associé à S_{16}, compatible avec la structure des générateurs choisis.

Afin de surmonter cette difficulté, on a utilisé le logiciel GAP (Groups, Algorithms, Programming), qui offre des outils puissants pour la manipulation algorithmique des groupes finis.

Concrètement, GAP a permis de :

- Générer des procédures opératoires couvrantes, en s'appuyant sur des algorithmes spécialisés pour l'énumération de sous-groupes de S_n ;
- Générer une liste de représentants pour les différents points de départ dans la partition en sous-groupes.

Grâce à cette approche, il a été possible d'implémenter la méthode sur S_{16}, en établissant une correspondance explicite entre l'ensemble des permutations et l'espace binaire restreint $C^*_{n'}$, tout en assurant la validité de la procédure opératoire choisie.

L'utilisation du logiciel GAP s'est révélée essentielle dès lors que l'on souhaite étendre la méthode à des groupes de grande taille ; GAP permet en effet de générer efficacement les trajectoires opératoires et les représentants nécessaires à la construction de l'énumération, ce qui constitue une étape fondamentale pour l'application de l'approche à des instances du TSP de dimension supérieure.

Cependant, un défaut majeur de cette approche est le manque de liberté dans le choix de l'ordre de la trajectoire opératoire et des représentants : les algorithmes de GAP imposent de n'avoir accès qu'à un seul type d'énumération, ce qui limite la flexibilité de la méthode. On ne peut donc pas sélectionner arbitrairement la structure de la trajectoire ou des représentants, ce qui peut s'avérer contraignant pour certaines applications où une organisation spécifique des états serait souhaitable.

Conclusion

Bilan

Ce rapport présente une étude approfondie sur la construction et l'implémentation de représentations opératoires pour l'énumération efficace des solutions de problèmes combinatoires, en particulier dans le contexte de l'informatique quantique hybride appliquée au problème du voyageur de commerce.

Le travail s'articule autour de plusieurs axes majeurs : la formalisation mathématique des représentations opératoires, la construction explicite d'isomorphismes entre espaces binaires restreints et groupes de permutations, l'analyse des partitions optimales pour l'encodage binaire minimal, et l'implémentation algorithmique sur des circuits quantiques simulés.

La Partie 2.1 établit les bases conceptuelles du formalisme des représentations opératoires et en expose la justification mathématique. Ce formalisme est présenté comme une proposition multidisciplinaire, visant à faciliter le transfert d'algorithmes et de méthodes entre différents domaines, notamment la théorie des groupes, la combinatoire, et l'algorithmique classique. La notion de procédure opératoire y est clarifiée, ainsi que la structure des espaces d'états et des générateurs, permettant de formaliser l'énumération des solutions combinatoires dans un cadre général. Cette approche offre un langage commun pour décrire et comparer des

stratégies d'optimisation et d'encodage, indépendamment du support (classique ou quantique), et ouvre la voie à une meilleure intégration des outils issus de la théorie des groupes, de la programmation, et de la physique quantique.

La Partie 2.2 propose une méthode systématique pour établir une correspondance entre l'espace des permutations S_n et un sous-ensemble binaire $C^*_{n'}$, en exploitant la structure arithmétique de $n!$ et la factorisation en blocs. Cette approche permet de minimiser le nombre de qubits nécessaires et d'optimiser la préparation des états quantiques, tout en garantissant une couverture exhaustive de l'espace des solutions. Les généralisations à des sous-groupes et à des partitions par orbites ouvrent la voie à une organisation structurée de l'espace d'états, facilitant l'exploration algorithmique et l'analyse du paysage de coût.

La Partie 3 détaille les résultats expérimentaux obtenus avec différents algorithmes maîtres (F-VQE, COBYLA) et architectures de circuits (p-layer ansatz, circuits censurés), sur des instances de tailles croissantes (S_4, S_8, S_{16}). Les performances observées confirment l'intérêt de la nouvelle méthode : réduction significative du gap moyen, meilleure concentration sur les solutions optimales.

Limitations

Malgré les avancées présentées dans ce travail, plusieurs limitations importantes doivent être soulignées :

- **Scalabilité et complexité algorithmique** : La construction explicite des procédures opératoires couvrantes (chemins hamiltoniens ou cycles) sur les graphes de Cayley associés à des groupes de grande taille, tels que S_{16} ou au-delà, demeure un défi majeur. La complexité combinatoire croît extrêmement rapidement avec n, rendant la génération exhaustive ou même partielle de telles trajectoires difficile, voire impossible, sans outils spécialisés comme GAP. Cette limitation restreint l'applicabilité pratique de la méthode à des instances de taille modérée.
- **Préparation quantique des états censurés** : La réalisation physique de circuits quantiques capables de censurer explicitement plusieurs états interdits (par exemple, exclure 11 états sur 16 pour S_{16}) est actuellement hors de portée des architectures quantiques disponibles. Les méthodes de préparation d'état existantes sont efficaces pour exclure un ou quelques états, mais deviennent rapidement inadaptées lorsque le ratio d'états interdits augmente. Cela limite la possibilité d'obtenir un encodage parfaitement compact pour des problèmes de grande taille.
- **Dépendance à la structure arithmétique** : L'efficacité de la partition optimale repose fortement sur la décomposition arithmétique de $n!$ et de $2^n - 1$. Dès que certains facteurs premiers de $n!$ ne figurent pas dans $2^n - 1$, le nombre de tuples interdits augmente, ce qui complexifie la construction et la préparation des états. Cette dépendance

structurelle impose des contraintes fortes sur le choix des partitions et sur la généralisabilité de la méthode à d'autres familles de problèmes combinatoires.

- **Flexibilité limitée dans le choix des trajectoires** : Comme souligné dans la Partie 3, l'utilisation d'outils comme GAP permet de générer des procédures opératoires couvrantes, mais impose une organisation spécifique des trajectoires et des représentants. Il n'est pas possible de sélectionner arbitrairement l'ordre des parcours ou la structure des orbites, ce qui peut être contraignant pour certaines applications où une organisation particulière des états serait souhaitée.
- **Expérimentation sur simulateur** : L'ensemble des résultats présentés repose sur des simulations numériques (Qiskit), et non sur des dispositifs quantiques réels. Les performances observées pourraient différer significativement en présence de bruit, de décohérence ou de limitations matérielles propres aux ordinateurs quantiques actuels. La transposition des méthodes proposées à des architectures physiques reste à valider expérimentalement.
- **Dépendance à l'algorithme maître** : Les performances de la méthode sont étroitement liées au choix de l'algorithme maître (F-VQE, COBYLA) et à ses paramètres (nombre de shots, profondeur du circuit, etc.). Certains phénomènes observés (comme l'avantage de la version p-layer dans certains cas) semblent spécifiques à la dynamique de l'algorithme utilisé, et leur généralité reste à confirmer.

En résumé, si la méthode proposée ouvre des perspectives prometteuses pour l'énumération et l'optimisation combinatoire en contexte quantique hybride, son extension à des instances de grande taille et sa mise en oeuvre sur des architectures physiques réelles nécessitent encore des avancées théoriques et techniques substantielles.

Perspectives

Les perspectives de recherche ouvertes par ce travail sont variées.

- Sur le plan théorique, l'étude approfondie des groupes finis, des graphes de Cayley et des structures combinatoires offre un cadre fertile pour généraliser la notion de représentation opératoire à d'autres familles de problèmes combinatoires. En particulier, l'utilisation des codes de Gray combinatoires [1] permet d'envisager la formalisation de nombreuses méthodes classiques sous le prisme des représentations opératoires. Cette approche ouvre la voie à une analyse systématique des algorithmes d'énumération et d'optimisation pour leur adaptation aux architectures quantiques hybrides, telles que les VQA présentées en Figure 1.
- Par ailleurs, le développement continu d'algorithmes quantiques pour la préparation et l'exploration d'états, souvent fondés sur des concepts issus de la théorie des groupes et des représentations, constitue un terrain privilégié pour l'application du formalisme

proposé. De nombreuses méthodes existantes peuvent être reformulées et étudiées sous l'angle des représentations opératoires, avec pour objectif de concevoir des algorithmes quantiques natifs, optimisés pour la structure des espaces combinatoires considérés.

Pour la suite de ce travail, il apparaît essentiel de s'intéresser plus précisément au choix et à la conception de l'algorithme maître M utilisé dans l'optimisation hybride. Les résultats obtenus montrent que la performance globale dépend fortement de la dynamique de l'algorithme maître, de ses paramètres, et de sa capacité à exploiter la structure opératoire induite par l'encodage.

Ce travail d'analyse et de design algorithmique fera l'objet de recherches complémentaires. L'objectif sera de déterminer quels types d'algorithmes maîtres (descente de gradient, méthodes bayésiennes, heuristiques combinatoires, etc.) sont les plus efficaces dans ce cadre, et d'étudier leur comportement en fonction de la taille du problème, de la profondeur du circuit, et du niveau de structuration de l'espace d'états.

Annexe

Notions de base en informatique quantique

En informatique quantique, un système est modélisé par un vecteur d'état $|\psi\rangle$ dans un espace de Hilbert $\mathcal{H}$.

La **notation de Dirac** [12], ou notation bra-ket, est couramment utilisée pour distinguer les vecteurs colonnes ($|\psi\rangle$, appelés "ket") des vecteurs lignes conjugués ($\langle\psi|$, appelés "bra"). Le **produit scalaire hermitien** entre deux états $|\phi\rangle$ et $|\psi\rangle$ est noté

$$\langle\phi|\psi\rangle,$$

et correspond essentiellement au produit d'un vecteur ligne (bra) $\langle\phi|$ par un vecteur colonne (ket) $|\psi\rangle$, tel que :

$$\langle\phi|\psi\rangle = \sum_{i=1}^{d} \overline{\phi_i}\psi_i$$

où $\overline{\phi_i}$ désigne le conjugué complexe du i-ème coefficient de ϕ, et d la dimension de l'espace de Hilbert considéré.

Définition : Un espace de Hilbert $\mathcal{H}$ est un espace vectoriel complexe muni d'un produit scalaire hermitien $\langle\cdot|\cdot\rangle$ vérifiant :

- Linéarité à gauche : $\langle ax + by|z\rangle = a\langle x|z\rangle + b\langle y|z\rangle$ pour $a, b \in \mathbb{C}$,
- Conjugaison à droite : $\langle x|y\rangle = \overline{\langle y|x\rangle}$,
- Positivité : $\langle x|x\rangle \geq 0$ et $\langle x|x\rangle = 0 \iff x = 0$.

Pour **un qubit**, l'espace de Hilbert est $\mathcal{H} = \mathbb{C}^2$. Pour un **registre de n qubits**, l'espace de Hilbert est le produit tensoriel de n copies de $\mathbb{C}^2$, noté

$$\mathcal{H} = \underbrace{\mathbb{C}^2 \otimes \cdots \otimes \mathbb{C}^2}_{n \text{ fois}} = (\mathbb{C}^2)^{\otimes n}.$$

Le produit tensoriel $\otimes$ est une opération bilinéaire qui, à deux espaces vectoriels V et W, associe un espace $V \otimes W$ dont les éléments sont des sommes finies de vecteurs de la forme $v \otimes w$ avec $v \in V$, $w \in W$, et qui vérifie la propriété de bilinéarité :

$$(av_1 + bv_2) \otimes w = a(v_1 \otimes w) + b(v_2 \otimes w), \qquad v \otimes (aw_1 + bw_2) = a(v \otimes w_1) + b(v \otimes w_2)$$

pour tous $a, b \in \mathbb{C}$, $v_1, v_2 \in V$, $w_1, w_2 \in W$.

Ainsi, pour n qubits, la dimension de l'espace de Hilbert est 2^n, et on a l'isomorphisme canonique

$$(\mathbb{C}^2)^{\otimes n} \simeq \mathbb{C}^{2^n}.$$

Exemple explicite : Soit deux états de base d'un qubit, $|0\rangle$ et $|1\rangle$, qui sont représentés par les vecteurs

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Le produit scalaire hermitien entre ces deux états s'écrit

$$\langle 0|1\rangle = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = 1 \times 0 + 0 \times 1 = 0,$$

et

$$\langle 0|0\rangle = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 1 \times 1 + 0 \times 0 = 1,$$

$$\langle 1|1\rangle = \begin{pmatrix} 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = 0 \times 0 + 1 \times 1 = 1.$$

Ainsi, les états de base vérifient les relations d'orthonormalité suivantes :

$$\langle 0|1\rangle = 0, \qquad \langle 0|0\rangle = \langle 1|1\rangle = 1.$$

Un état quantique général peut s'exprimer comme une **superposition linéaire** d'états de base orthonormés. Plus précisément, tout état $|\psi\rangle$ s'écrit sous la forme

$$|\psi\rangle = \sum_{x \in \{0,1\}^n} c_x |x\rangle,$$

où les coefficients $c_x \in \mathbb{C}$ sont des amplitudes complexes associées à chaque état de base $|x\rangle$, et satisfont la condition de **normalisation**

$$\sum_x |c_x|^2 = 1.$$

Cette propriété traduit la nature probabiliste de la mécanique quantique, chaque coefficient c_x étant associé à la probabilité $|c_x|^2$ d'observer l'état $|x\rangle$ lors d'une mesure.

Les opérations sur les états quantiques sont décrites par des **opérateurs unitaires**, qui modélisent les **portes quantiques** dans un circuit. Un opérateur unitaire U vérifie la relation fondamentale :

$$U^\dagger U = I,$$

où I désigne la matrice identité, et $U^\dagger$ désigne l'adjoint hermitien de U, défini comme la transposée conjuguée de U : si $U = (u_{ij})$, alors $U^\dagger = (\overline{u_{ji}})$, où $\overline{u_{ji}}$ est le conjugué complexe de u_{ji}.

Cette condition assure la préservation du produit scalaire, donc la conservation des probabilités lors de l'évolution du système.

Définition : Une matrice hermitienne est une matrice carrée M telle que $M = M^\dagger$.

Dans le contexte d'un circuit quantique, chaque porte correspond à un opérateur unitaire agissant sur une partie du registre : une porte **simple** agit sur un seul qubit (exemple : Hadamard, Pauli-X), tandis qu'une porte **multiple** agit sur plusieurs qubits simultanément (exemple : CNOT).

Illustration succincte, une présentation détaillée est fournie en Annexe :

- **Hadamard** : crée la superposition, agit sur un qubit.

  Matrice :
  $$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

  Action sur les états de base :
  $$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \qquad H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

- **Pauli-X** : équivalent à une inversion logique (NOT).

  Matrice :
  $$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

  Action sur les états de base :
  $$X|0\rangle = |1\rangle, \qquad X|1\rangle = |0\rangle$$

- **CNOT** : porte à deux qubits, réalise une opération conditionnelle (contrôle/intrication).

  Matrice (base ordonnée $|00\rangle, |01\rangle, |10\rangle, |11\rangle$) :
  $$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

  Action sur les états de base :
  $$\text{CNOT}|a, b\rangle = |a, b \oplus a\rangle$$

  où $a, b \in \{0, 1\}$ et $\oplus$ désigne la somme modulo 2.

L'étude statistique des états et des opérateurs quantiques fait intervenir la **mesure de Haar**, qui est l'unique mesure invariante [13] sur le groupe des matrices unitaires $U(d)$. Cette mesure, notée $d\mu_{\text{Haar}}$, permet de définir une notion d'intégration sur l'espace des états purs (de norme 1) ou des opérateurs, indépendamment du choix de la base. Autrement dit, pour toute fonction intégrable f et pour tout $V \in U(d)$, on a :

$$\int_{U(d)} f(U)\, d\mu_{\text{Haar}}(U) = \int_{U(d)} f(UV)\, d\mu_{\text{Haar}}(U) = \int_{U(d)} f(VU)\, d\mu_{\text{Haar}}(U).$$

De plus, la mesure de Haar est une mesure de probabilité. Par conséquent, l'intégrale d'une fonction $f(U)$ sur la mesure de Haar s'interprète comme l'espérance de $f(U)$ par rapport à la mesure de probabilité μ_{Haar}.

Définition : Pour chaque valeur propre m d'un opérateur hermitien M (c'est-à-dire un observable quantique, telle que l'énergie, le spin, ou toute grandeur mesurable représentée par une matrice hermitienne dans l'espace de Hilbert), soit E_m l'espace propre associé à m, de dimension d_m. On note $\{|\psi_{m,k}\rangle\}_{k=1}^{d_m}$ une base orthonormée de E_m. Le **projecteur** P_m sur E_m est alors défini par :

$$P_m = \sum_{k=1}^{d_m} |\psi_{m,k}\rangle\langle\psi_{m,k}|$$

où chaque terme $|\psi_{m,k}\rangle\langle\psi_{m,k}|$ est le projecteur sur la direction $|\psi_{m,k}\rangle$.

Définition : Si le système est dans l'état pur $|\psi\rangle$, la probabilité d'obtenir le résultat m lors de la mesure de M est donnée par :

$$p(m) = \langle\psi|P_m|\psi\rangle$$

Cas particulier : Si l'espace propre associé à m est de dimension 1, alors $P_m = |\psi_m\rangle\langle\psi_m|$, où $|\psi_m\rangle$ est l'état propre correspondant, et la probabilité devient :

$$p(m) = |\langle\psi_m|\psi\rangle|^2$$

Ainsi, la **valeur moyenne** (ou espérance) du résultat de la mesure de M sur l'état $|\psi\rangle$ s'exprime alors :

$$\mathbb{E}(M_{|\psi\rangle}) = \sum_m m\, p(m) = \sum_m m\, \langle\psi|P_m|\psi\rangle = \langle\psi|M|\psi\rangle.$$

Et donc, si l'on considère alors l'ensemble des états purs $|\psi\rangle$, la **valeur moyenne globale** de la mesure de M sur cet ensemble est donnée par :

$$\mathbb{E}_{|\psi\rangle}[\langle\psi|M|\psi\rangle] = \int_{|\psi\rangle\in\mathbb{C}^d,\, \|\psi\|=1} \langle\psi|M|\psi\rangle\, d\mu_{\text{Haar}}(|\psi\rangle).$$

Enfin, la **variance** de M sur l'ensemble des états purs s'exprime également en termes d'intégrales de Haar :

$$\mathrm{Var}_{|\psi\rangle}(M) = \int_{|\psi\rangle} \langle\psi|M^2|\psi\rangle \, d\mu_{\mathrm{Haar}}(|\psi\rangle) - \left(\int_{|\psi\rangle} \langle\psi|M|\psi\rangle \, d\mu_{\mathrm{Haar}}(|\psi\rangle) \right)^2 .$$

Enjeux et contraintes du calcul quantique

Le calcul quantique exploite des propriétés uniques des systèmes quantiques, telles que la superposition et l'intrication, pour réaliser des tâches computationnelles inaccessibles aux ordinateurs classiques. Toutefois, la mise en oeuvre de ces principes est soumise à des contraintes physiques et mathématiques qui limitent la performance et la fiabilité des algorithmes.

Définition 5: Un état composé de plusieurs qubits est dit intriqué s'il ne peut être exprimé comme un produit tensoriel d'états individuels, c'est-à-dire :

$$|\psi\rangle \neq |\psi_1\rangle \otimes |\psi_2\rangle \otimes \cdots \otimes |\psi_n\rangle.$$

L'intrication est une propriété fondamentale des systèmes quantiques, car elle permet l'apparition de corrélations globale entre les qubits : la mesure d'un qubit influence directement l'état des autres, ce qui rend la manipulation et le contrôle de l'information particulièrement complexe et puissant. Ces corrélations sont au coeur de la puissance du calcul quantique : elles permettent aux algorithmes quantiques d'explorer simultanément des configurations globales de l'espace de solutions

Par exemple, dans le cadre d'un problème d'optimisation combinatoire, il est possible de préparer un état initial qui encode une superposition uniforme sur toutes les solutions possibles :

$$|\psi_0\rangle = \frac{1}{\sqrt{|S|}} \sum_{s \in S} |s\rangle$$

où $|S|$ est le nombre total de solutions et $|s\rangle$ représente l'état quantique associé à la solution s. Cette superposition permet à l'algorithme quantique d'agir sur l'ensemble des solutions en parallèle, via des opérations unitaires et des mesures adaptées, et d'exploiter les corrélations induites par l'intrication pour amplifier la probabilité d'obtenir une solution optimale lors de la mesure finale.

La limitation fondamentale de la vitesse à laquelle de telles corrélations peuvent se propager dans un système quantique à interactions locales est décrite par la **borne de Lieb-Robinson** (voir l'annexe pour une explication approfondie) :

Cette borne fondamentale induit qu'une porte quantique ne peut pas influencer instantanément un qubit distant. L'effet de l'opération se propage à une vitesse maximale v_Φ, et hors de cette

zone d'influence, les corrélations décroissent exponentiellement. En pratique, cela impose deux contraintes majeures :

- Les opérations non locales (entre qubits éloignés) doivent être décomposées en séquences de portes locales, ce qui rallonge le temps de calcul.
- L'architecture physique doit minimiser les distances entre qubits interactifs, sous peine de voir l'information se dégrader avant même d'être traitée.

En résumé, on ne peut pas ignorer la géométrie du système : le placement des qubits et l'ordre des portes deviennent aussi critiques que le calcul lui-même.

Par ailleurs, la **profondeur des circuits quantiques** désigne le nombre total d'étapes (ou couches) successives de portes unitaires appliquées sur le registre de qubits, c'est-à-dire la longueur de la séquence d'opérations élémentaires qui compose le circuit.

Plus précisément, la profondeur correspond au nombre maximal d'opérations qui doivent être exécutées **en série** sur le circuit pour transformer l'état initial en l'état final. Cette profondeur est limitée par l'accumulation des erreurs lors de l'application successive de portes unitaires.

En 2018, le rapport [14] indique que sur les meilleures plateformes, comme les ions piégés ou les circuits supraconducteurs, le taux d'erreur pour les portes à deux qubits dépasse souvent 0,1 % ,et qu'en pratique, avec ces machines bruitées, il est difficile d'exécuter des circuits comportant plus d'environ 1000 portes à deux qubits.

Barren Plateaus et Concentration de la Mesure

Le phénomène de Barren Plateaus désigne une difficulté majeure rencontrée dans l'optimisation des algorithmes variationnels quantiques, en particulier sur les architectures NISQ. Il apparaît lorsque l'on cherche à ajuster les paramètres d'un circuit quantique paramétré, typiquement dans le cadre d'une optimisation combinatoire.

On considère (Voir [13] partie 9 pour une démosntrations complète) un circuit quantique composé de n qubits, dont l'état final dépend d'un vecteur de paramètres réels $\theta = (\theta_1, \dots, \theta_p)$. L'objectif est de minimiser une fonction coût de la forme

$$C(\theta) = \langle \psi(\theta) | M | \psi(\theta) \rangle,$$

où $|\psi(\theta)\rangle$ est l'état quantique obtenu après application du circuit paramétré, et M est l'observable associé au problème à résoudre. (Pour rappelle, cette quantité correspond a la **valeur moyenne** du résultat de la mesure de M sur l'état $|\psi(\theta)\rangle$)

D'après les inégalités de concentration (voir Eq. (269) [13]), en concidérant un état $|\phi\rangle$ dans l'espace associé, alors pour tout états $|\psi\rangle$ tiré uniformément (mesure de Haar), la probabilité que $|\langle \psi | \phi \rangle|^2$ soit significativement différente de zéro décroît exponentiellement avec le nombre de qubits n.

Cela implique que les gradients $\frac{\partial}{\partial \theta} C(\theta)$ deviennent exponentiellement petits en moyenne :

$$\mathbb{E}\left[\left| \frac{\partial}{\partial \theta} C(\theta) \right|^2 \right] \sim O\left(\frac{1}{2^n} \right).$$

En conclusion, les architectures physiques des ordinateurs quantiques sont soumises à des contraintes majeures, tant sur le plan théorique que pratique. Ces limitations, concernant la taille des registres et la fiabilité des opérations, incitent à développer des algorithmes optimisés, capables d'exécuter les tâches souhaitées avec un nombre minimal de portes et de qubits.

Détail des portes

- **Hadamard** : crée la superposition, agit sur un qubit.

  Matrice :

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

  Détail du calcul de l'action sur les états de base :

  Pour $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$:

  $$H|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \times 1 + 1 \times 0 \\ 1 \times 1 + (-1) \times 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

  Pour $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$:

  $$H|1\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \times 0 + 1 \times 1 \\ 1 \times 0 + (-1) \times 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

- **Pauli-X** : équivalent à une inversion logique (NOT).

  Matrice :

  $$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

  Détail du calcul de l'action sur les états de base :

  Pour $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$:

  $$X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \times 1 + 1 \times 0 \\ 1 \times 1 + 0 \times 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$$

  Pour $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$:

  $$X|1\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \times 0 + 1 \times 1 \\ 1 \times 0 + 0 \times 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle$$

- **CNOT** : porte à deux qubits, réalise une opération conditionnelle (contrôle/intrication).

Matrice (base ordonnée $|00\rangle, |01\rangle, |10\rangle, |11\rangle$) :

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Détail du calcul de l'action sur les états de base :

Pour $|00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$:

$$\text{CNOT}|00\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = |00\rangle$$

Pour $|01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$:

$$\text{CNOT}|01\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = |01\rangle$$

Pour $|10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$:

$$\text{CNOT}|10\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = |11\rangle$$

Pour $|11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$:

$$\text{CNOT}|11\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = |10\rangle$$

Lieb-Robinson

Le papier d'origine [15], présente un résultat fondamental sur la vitesse de propagation des effets physiques dans les systèmes de spins quantiques.

Soit deux observables locales A et B agissant respectivement sur des régions disjointes du réseau, séparées par une distance d. On considère l'évolution temporelle de A, notée $\tau_t(A)$.

La borne s'énonce alors comme suit : il existe une constante de vitesse $v_\Phi > 0$, dépendant de la structure des interactions locales Φ, et des constantes $C, \mu > 0$, telles que pour tout $t \geq 0$,

$$\|[\tau_t(A), B]\| \leq C\|A\|\|B\| \exp\left(-\mu\left(d - v_\Phi t\right)\right)$$

où $[\cdot, \cdot]$ désigne le commutateur, et $\|\cdot\|$ la norme d'opérateur.

Autrement dit, l'influence dynamique de A sur B demeure négligeable tant que $d > v_\Phi t$: la propagation des corrélations est confinée à une région dite "cône de lumière quantique", d'ouverture v_Φ, analogue au cône de lumière relativiste. En dehors de cette zone d'influence, les effets sont exponentiellement supprimés.

Le commutateur $\|[\tau_t(A), B]\|$ quantifie la capacité d'information à se transmettre entre les deux régions : lorsque $d \gg v_\Phi t$, on a

$$\|[\tau_t(A), B]\| \sim \exp\left(-\mu(d - v_\Phi t)\right)$$

ce qui implique une décroissance exponentielle de la corrélation avec la distance et le temps, hors du cône causal.

Preuve proposition 2

$$
\begin{aligned}
(4,3,1,2) &\xrightarrow{(3,4)} (4,3,2,1)\\
&\xrightarrow{(1,2)} (3,4,2,1)\\
&\xrightarrow{(2,3)} (3,2,4,1)\\
&\xrightarrow{(3,4)} (3,2,1,4)\\
&\xrightarrow{(1,2)} (2,3,1,4)\\
&\xrightarrow{(3,4)} (2,3,4,1)\\
&\xrightarrow{(2,3)} (2,4,3,1)\\
&\xrightarrow{(1,2)} (4,2,3,1)\\
&\xrightarrow{(3,4)} (4,2,1,3)\\
&\xrightarrow{(1,2)} (2,4,1,3)\\
&\xrightarrow{(2,3)} (2,1,4,3)\\
&\xrightarrow{(3,4)} (2,1,3,4)\\
&\xrightarrow{(1,2)} (1,2,3,4)\\
&\xrightarrow{(3,4)} (1,2,4,3)\\
&\xrightarrow{(2,3)} (1,4,2,3)\\
&\xrightarrow{(1,2)} (4,1,2,3)\\
&\xrightarrow{(3,4)} (4,1,3,2)\\
&\xrightarrow{(1,2)} (1,4,3,2)\\
&\xrightarrow{(2,3)} (1,3,4,2)\\
&\xrightarrow{(3,4)} (1,3,2,4)\\
&\xrightarrow{(1,2)} (3,1,2,4)\\
&\xrightarrow{(3,4)} (3,1,4,2)\\
&\xrightarrow{(2,3)} (3,4,1,2)\\
&\xrightarrow{(1,2)} (4,3,1,2)
\end{aligned}
$$

On vérifie que la tajectoire opératoire décrite ci-dessus produit exactement la même suite de permutations que celle générée par l'algorithme de Trotter-Johnson ;

```python
def trotter_johnson(n):
    if n == 1:
        return [[1]]
    else:
        result = []
        prev = trotter_johnson(n - 1)
        for idx, perm in enumerate(prev):
            if idx % 2 == 0:
                # Insérer n de gauche à droite
                for i in range(len(perm) + 1):
                    new_perm = perm[:i] + [n] + perm[i:]
                    result.append(new_perm)
            else:
                # Insérer n de droite à gauche
                for i in reversed(range(len(perm) + 1)):
                    new_perm = perm[:i] + [n] + perm[i:]
                    result.append(new_perm)
        return result
```

```
[[4, 3, 2, 1],
 [3, 4, 2, 1],
 [3, 2, 4, 1],
 [3, 2, 1, 4],
 [2, 3, 1, 4],
 [2, 3, 4, 1],
 [2, 4, 3, 1],
 [4, 2, 3, 1],
 [4, 2, 1, 3],
 [2, 4, 1, 3],
 [2, 1, 4, 3],
 [2, 1, 3, 4],
 [1, 2, 3, 4],
 [1, 2, 4, 3],
 [1, 4, 2, 3],
 [4, 1, 2, 3],
 [4, 1, 3, 2],
 [1, 4, 3, 2],
 [1, 3, 4, 2],
 [1, 3, 2, 4],
 [3, 1, 2, 4],
 [3, 1, 4, 2],
 [3, 4, 1, 2],
 [4, 3, 1, 2]]
```

Preuve Theoreme 1

Les 16 cycles ont été obtenus par programmation exhaustive.

Les notations utilisées pour les motifs sont les suivantes :

- T_{12} : transposition $(1, 2)$
- T_{23} : transposition $(2, 3)$
- T_{34} : transposition $(3, 4)$

Les motifs sont donnés. Pour rappel, il faut répéter ces motifs 3 fois pour obtenir l'énumération complète des 24 éléments.

```
['T_12', 'T_34', 'T_23', 'T_12', 'T_34', 'T_12', 'T_23', 'T_34']

['T_34', 'T_12', 'T_34', 'T_23', 'T_12', 'T_34', 'T_12', 'T_23']

['T_34', 'T_23', 'T_12', 'T_34', 'T_12', 'T_23', 'T_34', 'T_12']

['T_12', 'T_23', 'T_34', 'T_12', 'T_34', 'T_23', 'T_12', 'T_34']

['T_12', 'T_34', 'T_12', 'T_23', 'T_34', 'T_12', 'T_34', 'T_23']

['T_34', 'T_23', 'T_34', 'T_23', 'T_12', 'T_23', 'T_12', 'T_23']

['T_23', 'T_34', 'T_23', 'T_34', 'T_23', 'T_12', 'T_23', 'T_12']

['T_12', 'T_23', 'T_34', 'T_23', 'T_34', 'T_23', 'T_12', 'T_23'] #W_S4 formé en Table 1

['T_23', 'T_34', 'T_23', 'T_12', 'T_23', 'T_12', 'T_23', 'T_34']

['T_23', 'T_34', 'T_12', 'T_34', 'T_23', 'T_12', 'T_34', 'T_12']

['T_12', 'T_23', 'T_12', 'T_23', 'T_34', 'T_23', 'T_34', 'T_23']

['T_34', 'T_12', 'T_23', 'T_34', 'T_12', 'T_34', 'T_23', 'T_12'] #TROTTER de la partie 2

['T_23', 'T_12', 'T_23', 'T_34', 'T_23', 'T_34', 'T_23', 'T_12']

['T_34', 'T_23', 'T_12', 'T_23', 'T_12', 'T_23', 'T_34', 'T_23']

['T_23', 'T_12', 'T_34', 'T_12', 'T_23', 'T_34', 'T_12', 'T_34']
```

```
['T_23', 'T_12', 'T_23', 'T_12', 'T_23', 'T_34', 'T_23', 'T_34']
```

Preuve Theoreme 2

Les 12 cycles ont été obtenus par programmation exhaustive.

A posteriori, il apparaît, comme indiqué dans [1] section 3.11, que le nombre de codes de Gray sur le cube est connu jusqu'à $n = 6$. Pour $n = 3$, il existe précisément 6 codes de Gray distincts, si l'on ne tient pas compte de l'orientation du cycle. Cela explique pourquoi 12 cycles on était trouvés : chaque cycle est toujours accompagné de son homologue obtenu par parcours dans le sens opposé.

Les notations utilisées pour les motifs sont les suivantes :

$$T_1 = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad T_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad T_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

Le point initial considéré est le vecteur :$(-1, -1, -1)$

```
['T_1', 'T_2', 'T_1', 'T_3', 'T_1', 'T_2', 'T_1', 'T_3']

['T_3', 'T_1', 'T_2', 'T_1', 'T_3', 'T_1', 'T_2', 'T_1']

['T_3', 'T_2', 'T_3', 'T_1', 'T_3', 'T_2', 'T_3', 'T_1'] #GRAY de la partie 2

['T_1', 'T_3', 'T_2', 'T_3', 'T_1', 'T_3', 'T_2', 'T_3']

['T_3', 'T_2', 'T_1', 'T_2', 'T_3', 'T_2', 'T_1', 'T_2']

['T_1', 'T_3', 'T_1', 'T_2', 'T_1', 'T_3', 'T_1', 'T_2']

['T_2', 'T_1', 'T_3', 'T_1', 'T_2', 'T_1', 'T_3', 'T_1']

['T_2', 'T_1', 'T_2', 'T_3', 'T_2', 'T_1', 'T_2', 'T_3']

['T_3', 'T_1', 'T_3', 'T_2', 'T_3', 'T_1', 'T_3', 'T_2']

['T_2', 'T_3', 'T_2', 'T_1', 'T_2', 'T_3', 'T_2', 'T_1']

['T_1', 'T_2', 'T_3', 'T_2', 'T_1', 'T_2', 'T_3', 'T_2']
```

```
['T_2', 'T_3', 'T_1', 'T_3', 'T_2', 'T_3', 'T_1', 'T_3']
```

Discussion sur la construction de la table 1.

La procédure $\mathcal{W}_{S_4}$ présentée en Table 1 est construite à partir du motif suivant :

$$\mathcal{W}_{C_3} = (\sigma_1,\ \sigma_2,\ \sigma_3,\ \sigma_2,\ \sigma_3,\ \sigma_2,\ \sigma_1,\ \sigma_2),$$

Une question particulièrement intéressante, d'un point de vue mathématique, consiste à comprendre pourquoi il n'est pas possible de définir une action opératoire π' sur C_3, fondée sur le produit matriciel à gauche et sur les inversions de bits strictes, qui permette à la procédure $\mathcal{W}_{C_3}$ d'induire une trajectoire opératoire à la fois couvrante et cyclique.

En effet, bien que l'on dispose de trois générateurs et que l'on puisse envisager trois inversions de bits à leur associer, une analyse exhaustive de toutes les attributions possibles montre qu'aucune ne conduit à une trajectoire opératoire sur C_3 qui soit à la fois couvrante et cyclique pour la procédure $\mathcal{W}_{C_3}$.

Pour expliciter ce phénomène, on peut dresser la liste exhaustive des attributions possibles des générateurs $\sigma_1, \sigma_2, \sigma_3$ à l'inversion des trois bits de C_3 :

Il existe $3! = 6$ façons d'associer chaque générateur à l'inversion d'un bit distinct. On note chaque attribution sous la forme $(\sigma_1 \to i_1,\ \sigma_2 \to i_2,\ \sigma_3 \to i_3)$, où $i_j \in \{1, 2, 3\}$ désigne le bit inversé par σ_j.

Voici la liste exhaustive :

1. $(\sigma_1 \to 1,\ \sigma_2 \to 2,\ \sigma_3 \to 3)$
2. $(\sigma_1 \to 1,\ \sigma_2 \to 3,\ \sigma_3 \to 2)$
3. $(\sigma_1 \to 2,\ \sigma_2 \to 1,\ \sigma_3 \to 3)$
4. $(\sigma_1 \to 2,\ \sigma_2 \to 3,\ \sigma_3 \to 1)$
5. $(\sigma_1 \to 3,\ \sigma_2 \to 1,\ \sigma_3 \to 2)$
6. $(\sigma_1 \to 3,\ \sigma_2 \to 2,\ \sigma_3 \to 1)$

Pour illustrer ce phénomène, on explicite ici les trajectoires induites par la procédure $\mathcal{W}_{C_3}$ pour chacune des six attributions possibles des générateurs aux inversions de bits. On montre que, dans chaque cas, la trajectoire n'est jamais à la fois couvrante et cyclique : soit elle ne visite pas tous les états, soit elle ne revient pas au point de départ.

On rappelle que la procédure opératoire considérée est :

$$\mathcal{W}_{C_3} = (\sigma_1,\ \sigma_2,\ \sigma_3,\ \sigma_2,\ \sigma_3,\ \sigma_2,\ \sigma_1,\ \sigma_2)$$

et que l'espace d'états est $C_3 = \{(0,0,0),\ (0,0,1),\ (0,1,0),\ (0,1,1),\ (1,0,0),\ (1,0,1),\ (1,1,0),\ (1,1,1)\}$.

On note $q = (q_1, q_2, q_3)$ l'état courant, et chaque générateur σ_i inverse le bit i.

Cas 1 : $(\sigma_1 \to 1,\ \sigma_2 \to 2,\ \sigma_3 \to 3)$

Point de départ : $q_0 = (0, 0, 0)$

1. σ_1 : $(1, 0, 0)$
2. σ_2 : $(1, 1, 0)$
3. σ_3 : $(1, 1, 1)$
4. σ_2 : $(1, 0, 1)$
5. σ_3 : $(1, 0, 0)$
6. σ_2 : $(1, 1, 0)$

Échec : à l'étape 6, la trajectoire repasse par un état déjà visité.

Cas 2 : $(\sigma_1 \to 1,\ \sigma_2 \to 3,\ \sigma_3 \to 2)$

Point de départ : $q_0 = (0, 0, 0)$

1. σ_1 : $(1, 0, 0)$
2. σ_2 : $(1, 0, 1)$
3. σ_3 : $(1, 1, 1)$
4. σ_2 : $(1, 1, 0)$
5. σ_3 : $(1, 0, 0)$
6. σ_2 : $(1, 0, 1)$

Échec : à l'étape 6, la trajectoire repasse par un état déjà visité.

Cas 3 : $(\sigma_1 \to 2,\ \sigma_2 \to 1,\ \sigma_3 \to 3)$

Point de départ : $q_0 = (0, 0, 0)$

1. σ_1 : $(0, 1, 0)$
2. σ_2 : $(1, 1, 0)$
3. σ_3 : $(1, 1, 1)$
4. σ_2 : $(0, 1, 1)$
5. σ_3 : $(0, 1, 0)$
6. σ_2 : $(1, 1, 0)$

Échec : à l'étape 6, la trajectoire repasse par un état déjà visité.

Cas 4 : $(\sigma_1 \to 2,\ \sigma_2 \to 3,\ \sigma_3 \to 1)$

Point de départ : $q_0 = (0, 0, 0)$

1. σ_1 : $(0, 1, 0)$
2. σ_2 : $(0, 1, 1)$
3. σ_3 : $(1, 1, 1)$
4. σ_2 : $(1, 1, 0)$

5. σ_3 : $(0,1,0)$
6. σ_2 : $(0,1,1)$

Échec : à l'étape 6, la trajectoire repasse par un état déjà visité.

Cas 5 : $(\sigma_1 \to 3,\ \sigma_2 \to 1,\ \sigma_3 \to 2)$

Point de départ : $q_0 = (0,0,0)$

1. σ_1 : $(0,0,1)$
2. σ_2 : $(1,0,1)$
3. σ_3 : $(1,1,1)$
4. σ_2 : $(0,1,1)$
5. σ_3 : $(0,0,1)$
6. σ_2 : $(1,0,1)$

Échec : à l'étape 6, la trajectoire repasse par un état déjà visité.

Cas 6 : $(\sigma_1 \to 3,\ \sigma_2 \to 2,\ \sigma_3 \to 1)$

Point de départ : $q_0 = (0,0,0)$

1. σ_1 : $(0,0,1)$
2. σ_2 : $(0,1,1)$
3. σ_3 : $(1,1,1)$
4. σ_2 : $(1,0,1)$
5. σ_3 : $(0,0,1)$
6. σ_2 : $(1,1,1)$

Échec : à l'étape 6, la trajectoire repasse par un état déjà visité.

Conclusion :

Dans tous les cas, la procédure $\mathcal{W}_{C_3}$, quelle que soit l'attribution des générateurs aux bits, ne permet pas d'énumérer exhaustivement les 8 états de C_3 : la trajectoire obtenue n'est jamais couvrante.

Ce constat a était une des motivations pour introduire le formalisme des représentations opératoires : dans le cas de la représentation de S_4 présentée en Table 1, il a été indispensable de sélectionner un ensemble spécifique de matrices pour définir π', de manière à garantir que le motif W_{C_3} induise effectivement une trajectoire couvrante sur l'espace d'états considéré.

Il apparaît que la nature du groupe générateur joue un rôle déterminant, en particulier le degré des générateurs considérés : les inversions strictes correspondent à des involutions (d'ordre 2), tandis que, dans la construction de la Table 1, on utilise notamment deux copies de la matrice

$$\begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

qui est d'ordre 4.

Ce constat met en évidence une propriété structurelle intéressante : il n'existe aucun groupe agissant sur $E' = \{-1, 1\}^3$ par multiplication matricielle à gauche, engendré par trois générateurs d'ordre 2, qui admette une trajectoire opératoire couvrante pour la procédure $\mathcal{W}_{C_3}$.

Informations complémentaire sur G_{384}

Les générateurs du sous-groupe G_{384} utilisé sont les suivants :

- $(1\,4)(2\,3)$
- $(1\,2)$
- $(5\,8)(6\,7)$
- $(5\,6)$
- $(3\,6)(4\,5)$
- $(3\,4)$

Chacun de ces générateurs correspond à une permutation cyclique sur les indices indiqués, et leur composition engendre l'ensemble du sous-groupe G_{384} de S_8.

La procédure opératoire suit une règle de répétition structurée : elle consiste en un motif de taille 24, où les 23 premiers générateurs sont identiques, et seul le 24ème diffère.

Étape	Séquence W des générateurs appliqué
1	(1,4)(2,3)
2	(5,8)(6,7)
3	(1,4)(2,3)
4	(3,6)(4,5)
5	(1,4)(2,3)
6	(5,8)(6,7)
7	(1,4)(2,3)
8	(3,6)(4,5)
9	(5,8)(6,7)
10	(1,4)(2,3)
11	(5,8)(6,7)
12	(3,6)(4,5)
13	(1,4)(2,3)
14	(5,8)(6,7)
15	(1,4)(2,3)
16	(3,6)(4,5)
17	(5,8)(6,7)
18	(1,4)(2,3)
19	(5,8)(6,7)
20	(3,6)(4,5)
21	(1,4)(2,3)
22	(5,8)(6,7)
23	(1,4)(2,3)

La procédure totale, en notant W ce motif de base de taille 23, s'écrit sous forme de séquence:

$$
\begin{bmatrix}
W,\ (1,2),\ W,\ (3,4),\ W,\ (5,6),\ W,\ (5,6),\ W,\ (5,6),\ W,\ (3,4),\ W,\ (5,6),\ W,\ (5,6),\ W,\\
(1,2),\ W,\ (3,4),\ W,\ (5,6),\ W,\ (5,6),\ W,\ (5,6),\ W,\ (3,4),\ W,\ (5,6),\ W,\ (5,6)
\end{bmatrix}
$$

La liste exhaustive des 105 représentants sélectionnés, correspondant aux entêtes des 105 colonnes du tableau opératoire (c'est-à-dire aux points de départ de la trajectoire opératoire décrite précédemment), est présentée ci-dessous dans l'ordre utilisé pour la construction de l'énumération. Chaque représentant définit une classe d'équivalence distincte sous l'action du sous-groupe G_{384}, assurant ainsi la couverture complète de l'espace des orbites pour la procédure opératoire considérée.

```
Représentants
1,2,3,4,5,6,7,8
```

```
1,2,3,4,5,7,6,8
1,2,3,4,6,7,5,8
1,2,3,5,6,4,7,8
1,2,3,5,7,4,6,8
1,2,3,5,8,4,6,7
1,2,4,5,8,7,6,3
1,2,4,7,8,5,6,3
1,2,5,7,8,4,6,3
1,2,5,4,8,6,7,3
1,2,5,8,4,6,7,3
1,2,5,6,4,8,7,3
1,2,7,6,4,5,8,3
1,2,7,5,4,6,8,3
1,2,7,4,5,6,8,3
1,3,2,4,5,6,8,7
1,3,2,4,5,7,8,6
1,3,2,4,6,7,8,5
1,3,2,5,6,4,8,7
1,3,2,5,6,8,4,7
1,3,2,5,6,7,4,8
1,3,2,6,8,7,4,5
1,3,2,6,8,5,4,7
1,3,2,6,8,4,5,7
1,3,2,7,8,6,5,4
1,3,2,7,4,6,5,8
1,3,2,7,5,6,4,8
1,3,6,7,5,4,2,8
1,3,5,7,6,4,2,8
1,3,4,7,6,5,2,8
1,4,8,7,6,5,2,3
1,4,8,6,7,5,2,3
1,4,7,6,8,5,2,3
1,4,3,6,8,7,2,5
1,4,3,7,8,6,2,5
1,4,6,7,8,3,2,5
1,4,6,2,8,7,3,5
1,4,6,2,8,5,3,7
1,4,6,2,8,3,5,7
1,4,7,2,8,6,5,3
1,4,7,2,3,6,5,8
1,4,7,2,5,6,3,8
1,4,8,2,7,6,3,5
```

```
1,4,8,2,7,5,3,6
1,4,8,2,7,3,5,6
6,4,8,7,2,3,5,1
7,4,8,6,2,3,5,1
7,6,8,4,2,3,5,1
3,6,8,7,2,4,5,1
3,7,8,6,2,4,5,1
3,8,7,6,2,4,5,1
7,8,2,6,3,4,5,1
7,3,2,6,8,4,5,1
8,3,2,6,7,4,5,1
4,3,8,6,7,2,5,1
4,8,3,6,7,2,5,1
3,8,4,6,7,2,5,1
2,8,4,3,7,6,5,1
2,8,6,3,7,4,5,1
2,8,6,4,7,3,5,1
7,8,5,4,2,3,6,1
5,8,7,4,2,3,6,1
4,8,7,5,2,3,6,1
4,2,7,8,5,3,6,1
4,2,5,8,7,3,6,1
4,2,3,8,7,5,6,1
4,3,7,8,2,5,6,1
4,8,7,3,2,5,6,1
3,8,7,4,2,5,6,1
3,4,7,2,8,5,6,1
3,5,7,2,8,4,6,1
4,5,7,2,8,3,6,1
7,5,8,2,4,3,6,1
7,4,8,2,5,3,6,1
5,4,8,2,7,3,6,1
5,4,8,6,2,3,7,1
6,4,8,5,2,3,7,1
8,4,6,5,2,3,7,1
2,4,6,8,5,3,7,1
2,4,6,3,5,8,7,1
2,4,6,5,3,8,7,1
2,5,6,8,3,4,7,1
2,5,6,3,8,4,7,1
2,5,8,3,6,4,7,1
8,5,4,3,6,2,7,1
```

```
3,5,4,8,6,2,7,1
4,5,3,8,6,2,7,1
6,5,3,4,8,2,7,1
6,4,3,5,8,2,7,1
5,4,3,6,8,2,7,1
5,4,7,6,3,2,8,1
5,7,4,6,3,2,8,1
5,6,4,7,3,2,8,1
7,6,4,2,3,5,8,1
7,5,4,2,3,6,8,1
6,5,4,2,3,7,8,1
2,5,4,3,6,7,8,1
2,5,4,7,6,3,8,1
2,5,3,7,6,4,8,1
2,6,5,7,3,4,8,1
2,6,4,7,3,5,8,1
2,6,3,7,4,5,8,1
5,6,2,7,4,3,8,1
4,6,2,7,5,3,8,1
4,5,2,7,6,3,8,1
```

p-layer-ansatz

Le concept de **p-layer ansatz** désigne une architecture de circuit quantique paramétré composée de p couches successives d'opérations unitaires. Chaque couche (layer) applique une séquence de portes quantiques (typiquement des rotations locales et des portes d'intrication comme CNOT) sur l'ensemble des qubits, selon des paramètres libres optimisés par l'algorithme maître. L'objectif est de créer une superposition contrôlée des états quantiques, permettant d'explorer efficacement l'espace des solutions du problème combinatoire.

Dans le cadre du p-layer ansatz, la profondeur p du circuit correspond au nombre de répétitions du motif de base : plus p est élevé, plus le circuit est expressif et capable de représenter des distributions complexes sur l'espace d'états. Cette architecture est largement utilisée dans les algorithmes hybrides quantiques- classiques (comme VQE ou QAOA), car elle offre un compromis entre simplicité de mise en œuvre et capacité d'approximation des états optimaux.

En pratique, le schéma p-layer ansatz consiste à alterner, pour chaque couche, des opérations locales (rotations sur chaque qubit) et des opérations d'intrication (CNOT ou autres portes à deux qubits), selon une topologie choisie (par exemple, linéaire ou en anneau). Les paramètres de chaque porte sont ajustés au fil des itérations pour minimiser une fonction de coût associée au problème traité.

Ici, on a choisi p = 1, ce qui correspond au circuit suivant :

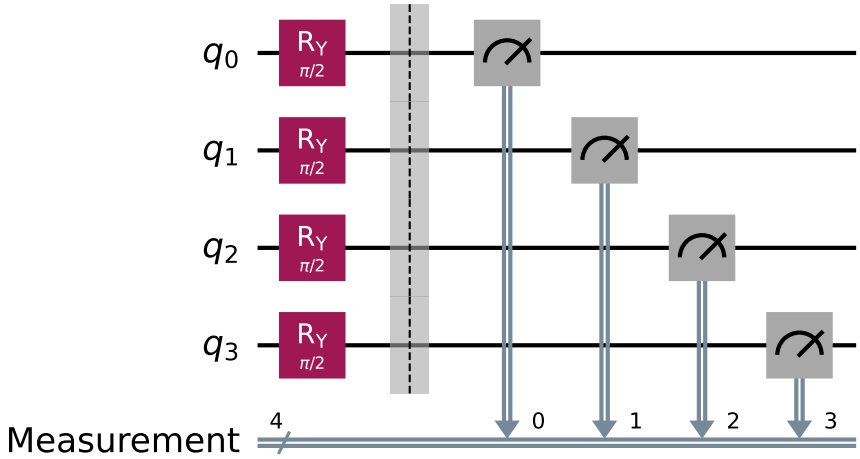

Figure 18: 1-layer-ansatz

Données complémentaires, expériences additionnelles, code et ressources

L'ensemble des données complémentaires relatives aux différents sous-groupes étudiés, aux analyses expérimentales approfondies, ainsi qu'aux codes sources et jeux de données générés dans le cadre de ce travail, sont mis à disposition sur un dépôt Git dédié. Cette organisation s'impose en raison du volume conséquent des ressources produites, qui excède largement les capacités de présentation dans le manuscrit. Le dépôt regroupe notamment :

- les listings exhaustifs des générateurs et représentants utilisés pour chaque sous-groupe testé ;
- les scripts d'implémentation des procédures opératoires et des algorithmes d'optimisation ;
- les résultats détaillés des expérimentations numériques (courbes, histogrammes, matrices de coûts, etc.) ;
- les visualisations graphiques des structures opératoires et des graphes de Cayley associés ;
- toute documentation technique complémentaire nécessaire à la reproduction et à l'analyse des expériences.

LIEN : https://github.com/GagginiLorenzo/Representations-Operatoire

1. Mütze T (2024) Combinatorial gray codes-an updated survey

2. Schupp PE (1998) On the structure of hamiltonian cycles in cayley graphs of finite quotients of the modular group. Theoretical Computer Science 204(1):233–248. https://doi.org/https://doi.org/10.1016/S0304-3975(98)00041-3

3. Kowalski E (2017) Representation theory

4. Grover LK (1996) A fast quantum mechanical algorithm for database search

5. Droste S, Jansen T, Wegener I (2004) Upper and lower bounds for randomized search heuristics in black-box optimization

6. Johnson SM (1963) Generation of permutations by adjacent transposition. Mathematics of Computation 17(83):282–285

7. Commons W (2020) File:symmetric group 4; cayley graph 1,2,6 (3D); steinhaus–johnson–trotter.svg — wikimedia commons, the free media repository

8. Pak I, Radoičić R (2009) Hamiltonian paths in cayley graphs. Discrete Mathematics 309(17):5501–5508. https://doi.org/https://doi.org/10.1016/j.disc.2009.02.018

9. Shukla A, Vedula P (2024) An efficient quantum algorithm for preparation of uniform quantum superposition states. Quantum Information Processing 23(2). https://doi.org/10.1007/s11128-024-04258-4

10. Cayley A (1854) VII. On the theory of groups, as depending on the symbolic equation n =1. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 7(42):40–47. https://doi.org/10.1080/14786445408647421

11. Meghazi I, Bourreau E (2025) Encodage indirect amélioré pour résolution du problème du voyageur de commerce avec algorithmes variationnels quantiques

12. Dirac PAM (1930) The principle of superposition. In: The principles of quantum mechanics, 4th ed. Oxford University Press, Oxford

13. Mele AA (2024) Introduction to haar measure tools in quantum information: A beginner's tutorial. Quantum 8:1340. https://doi.org/10.22331/q-2024-05-08-1340

14. Preskill J (2018) Quantum computing in the NISQ era and beyond. Quantum 2:79. https://doi.org/10.22331/q-2018-08-06-79

15. Lieb EH, Robinson DW (2004) The finite group velocity of quantum spin systems. In: Nachtergaele B, Solovej JP, Yngvason J (eds) Statistical mechanics: Selecta of elliott h. lieb. Springer Berlin Heidelberg, Berlin, Heidelberg, pp 425–431