

Міністерство освіти і науки України
Державний університет «Житомирська політехніка»

Кваліфікаційна наукова
праця на правах рукопису

САПОЖНИК ДМИТРО ОЛЕГОВИЧ

УДК 004.383:[530.145:004.021]

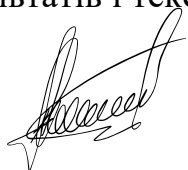
ДИСЕРТАЦІЯ
**МЕТОДИ ТА АЛГОРИТМИ ДЛЯ РОЗВ’ЯЗАННЯ ЗАДАЧІ
МАКСИМАЛЬНОГО РОЗРІЗУ ЗА ДОПОМОГОЮ КВАНТОВО-
ГІБРИДНИХ АЛГОРИТМІВ**

121 Інженерія програмного забезпечення

12 Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело



/Д. О. Сапожник

Науковий керівник Морозов Андрій Васильович, кандидат технічних наук,
доцент кафедри інженерії програмного забезпечення

Житомир – 2026

АНОТАЦІЯ

Сапожник Д. О. Методи та алгоритми для розв’язання задачі максимального розрізу за допомогою квантово-гібридних алгоритмів. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії з галузі знань 12 Інформаційні технології за спеціальністю 121 Інженерія програмного забезпечення. – Державний університет «Житомирська політехніка» Міністерства освіти і науки України, Житомир, 2026.

Задача максимального розрізу графа (Max-Cut) належить до класу NP-складних задач комбінаторної оптимізації та займає важливе місце в теорії алгоритмів, теорії складності та прикладній математиці. Мета задачі полягає у знаходженні такого розбиття множини вершин неорієнтованого графа на дві підмножини, при якому кількість або сумарна вага ребер між цими підмножинами є максимальною. Незважаючи на простоту формулювання, задача характеризується експоненційним зростанням простору можливих рішень зі збільшенням кількості вершин, що обумовлює її високу обчислювальну складність.

Задача Max-Cut застосовується в задачах оптимізації топології телекомунікаційних і енергетичних мереж, балансування навантаження в розподілених системах, кластеризації даних, аналізі мережевих структур, фінансовому моделюванні, а також у задачах проєктування інтегральних схем і транспортному плануванні.

Класичні підходи до розв’язання Max-Cut демонструють суттєві обмеження при зростанні розмірності графа. Точні алгоритми, зокрема метод повного перебору та метод гілок і меж, забезпечують знаходження глобально оптимального розв’язку, однак їх експоненційна часова складність обмежує їх використання для задач середньої та великої розмірності. Апроксимаційні методи, зокрема алгоритм Гоєманса-Вільямсона (Goemans-Williamson), забезпечують теоретичні гарантії якості розв’язку, проте пов’язані зі значними

обчислювальними витратами, зумовленими використанням напіввизначеного програмування. Евристичні та метаевристичні алгоритми, такі як локальний пошук, генетичні алгоритми та імітація відпалу, мають кращу масштабованість, однак не гарантують досягнення глобального оптимуму та характеризуються варіативністю результатів.

У контексті розвитку квантових обчислень задача пошуку ефективних методів розв'язання NP-складних задач набуває нового значення. Поява варіаційних квантових алгоритмів, зокрема Quantum Approximate Optimization Algorithm (QAOA), створила передумови для використання квантових ресурсів у задачах комбінаторної оптимізації. Разом з тим, сучасний стан квантових технологій визначається характеристиками NISQ-ери (Noisy Intermediate-Scale Quantum), яка обмежується невеликою кількістю кубітів, обмеженою глибиною квантових схем, а також впливом шуму та декогеренції. У таких умовах реалізація складних параметризованих квантових схем і багатоітераційних гібридних алгоритмів є ускладненою.

Додатковим обмеженням варіаційних квантових алгоритмів є необхідність багаторазової взаємодії між квантовим і класичним компонентами для оптимізації параметрів квантової схеми. Така ітераційна процедура супроводжується накладними витратами, підвищеною чутливістю до шуму та складністю оптимізаційного ландшафту, що може негативно впливати на ефективність їх застосування в умовах реальних квантових пристроїв.

У зв'язку з цим актуальність теми дисертаційного дослідження зумовлена необхідністю розробки квантово-гібридних алгоритмів розв'язання задачі максимального розрізу графа, орієнтованих на зменшення глибини квантових схем, скорочення кількості ітерацій класичної оптимізації та підвищення стійкості до шуму в умовах сучасних квантових обчислювальних систем.

Метою дисертаційного дослідження є розробка методів та алгоритмів для розв'язання задачі максимального розрізу за допомогою квантово-гібридних алгоритмів.

Зміст дисертаційної роботи складається з трьох розділів, у яких наведено та обґрунтовано основні результати проведеного дослідження.

У **вступі** обґрунтовано актуальність теми дослідження, визначено об'єкт, предмет, мету та завдання, сформульовано наукову новизну, практичну значущість одержаних результатів, представлено методи розробки які були використані для отримання результатів, зазначено інформацію про публікацію та апробацію результатів дисертаційного дослідження.

У **першому розділі** здійснено ґрунтовний аналіз поточного стану розвитку квантових технологій, а саме проведено аналіз публікацій за останні 10 років у науково метричній базі даних Scopus, що показало значне збільшення інтересу до квантових технологій у сфері комп'ютерних наук. У сфері інвестицій також помітне значне підвищення фінансування, що корелюється з підвищенням зацікавленості науковців, яке виражене через публікацію наукових робіт. Проаналізовано діяльність провідних компаній, що здійснюють дослідження у сфері квантових технологій, визначено основні напрями їх розвитку та підходи до реалізації квантових обчислювальних систем.

Досліджено топологічні особливості фізичної реалізації кубітів і визначено основні обмеження, що виникають при розробці квантових комп'ютерів. Наведено формальний опис задачі максимального розрізу та виконано аналіз класичних і квантово-гібридних алгоритмів її розв'язання. Окрему увагу приділено дослідженню одно- та багатокубітних квантових вентилів, що є необхідною основою для побудови квантово-гібридних алгоритмів.

Другий розділ присвячено розробці квантово-гібридного алгоритму QASPA. У межах розділу виконано аналіз багатоітераційної схеми роботи алгоритму QAOA та запропоновано одноітераційний підхід, що дозволяє

зменшити кількість звернень до квантового компонента алгоритму. Розроблено модель взаємодії основних компонентів алгоритму QASPA та визначено функціональну структуру кожного з них. Класична частина алгоритму реалізує формалізацію задачі максимального розрізу у вигляді матриці кутів, що використовується для параметризації квантової схеми.

Проведено дослідження вибору параметрів, за результатами якого встановлено, що використання значення кута $\pi/4$ забезпечує стабільні результати при розв'язанні задачі. Запропоновано метод масштабування кількості запусків квантової схеми, який дозволяє визначати достатню кількість вимірювань для виділення найбільш імовірних розв'язків на фоні шуму. Розроблено метод побудови квантової схеми, що базується на використанні шару гейтів Адамара та послідовності операцій CX-RY-CX, які забезпечують кодування структури графа у квантовому стані. Проведено математичний аналіз запропонованої схеми, який підтверджує коректність її побудови. Також описано завершальний етап алгоритму, що передбачає аналіз результатів квантових вимірювань та визначення оптимального розв'язку.

У **третьому розділі** дисертаційного дослідження виконано аналіз сучасних платформ для розроблення квантових алгоритмів та реалізовано алгоритм QASPA на базі платформи Qiskit. Проведено експериментальні запуски алгоритму на квантових симуляторах і реальних квантових пристроях, що підтвердило коректність його реалізації.

Для оцінювання якості отриманих результатів проведено серію експериментів на тестових графах малої та середньої розмірності, а також на автоматично згенерованих графах із кількістю вершин від 3 до 14 різних топологій, зокрема повних, циклічних і випадкових графів моделі Erdős-Rényi з різними коефіцієнтами зв'язності. За результатами експериментів встановлено, що алгоритм QASPA забезпечує отримання оптимальних або близьких до оптимальних розв'язків із середнім коефіцієнтом апроксимації в межах 0.98-1.00, при цьому в усіх випадках значення коефіцієнта перевищувало порогове значення $p > 0.9$. Еталонні значення було отримано

за допомогою методу повного перебору. Додатково виконано порівняльний аналіз із алгоритмом QAOA, результати якого демонструють більшу варіативність та зниження якості розв'язків при збільшенні розмірності задачі.

Алгоритм QASPA також оцінено за показником успішності, який становив 100% для всіх тестових випадків незалежно від типу графа та його розмірності в досліджуваному діапазоні. Окремо проведено експериментальне дослідження часових характеристик алгоритму на повних графах із кількістю вершин до 19. Встановлено, що час виконання алгоритму знаходиться в межах 0.0846-0.4643 секунд та не демонструє експоненційного зростання зі збільшенням розмірності задачі, що свідчить про стабільність його обчислювальних характеристик у дослідженому діапазоні.

У результаті проведених у межах дисертаційного дослідження наукових досліджень **вперше** отримано такі **наукові результати**:

- розроблено квантово-гібридний алгоритм QASPA для розв'язання задачі максимального розрізу графа, який, на відміну від алгоритму QAOA, реалізує одноітераційну архітектуру без використання багатоітераційної класичної оптимізації параметрів, що дозволяє зменшити обчислювальні витрати та забезпечує стабільність результатів із досягненням 100% успішності виконання алгоритму для досліджених випадків при середньому коефіцієнті апроксимації в межах 0.98-1.00;

- запропоновано детермінований метод графо-залежної параметризації варіаційної квантової схеми, що базується на відображенні структури графа у матрицю кутів обертання з фіксованим значенням $\pi/4$ для наявних ребер та реалізує фазове кодування зв'язків графа через обертання навколо осі O_y ;

- розроблено метод масштабування кількості квантових запусків на основі афінно-експоненційної апроксимації, який дозволяє визначати статистично достатню кількість вимірювань для ідентифікації оптимального розв'язку при зростанні розмірності задачі;

- запропоновано метод побудови квантових схем для задач комбінаторної оптимізації, що передбачає використання шару гейтів Адамара

та тригетового блоку CX-RY-CX для кодування ребер графа та забезпечує зменшення глибини квантового кола і адаптацію до обмежень сучасних квантових процесорів.

Практичне значення результатів роботи полягає у:

- розробці програмних рішень для розв’язання задач комбінаторної оптимізації на основі квантово-гібридного алгоритму QASPA для оптимізації логістичних маршрутів, балансування навантаження в мережевих системах та задачах розподілу ресурсів;

- створенні прикладних програмних модулів для інтеграції з платформами квантових обчислень (зокрема IBM Quantum) для практичної апробації та використання алгоритму на симуляторах і реальних квантових процесорах;

- розробці методичного підходу до перетворення прикладних задач у графові моделі типу Max-Cut та їх розв’язання за допомогою квантово-гібридного алгоритму QASPA, що включає правила побудови квантових схем, визначення параметрів та оцінювання результатів і може бути використаний у системах підтримки прийняття рішень.

Впровадження результатів дослідження в практику підтверджується довідками у ТОВ «Альянс Діджитал», у компанії ТОВ «УКРАЇНСЬКИЙ ЦЕНТР ДУАЛЬНОЇ ОСВІТИ» та у компанії ТОВ «Н-ІКС ДЕЛІВЕРІ» (довідки додаються).

За результатами дисертаційних досліджень було опубліковано 11 наукових праць, з них 4 статей у фахових наукових виданнях України категорії «Б», 1 стаття опублікована у матеріалах міжнародної конференції з індексацією у базі даних Scopus та 6 у матеріалах наукових конференцій.

Ключові слова: квантові обчислення, гібридні алгоритми, алгоритми оптимізації, комбінаторна оптимізація, задача максимального розрізу, NP-складні задачі, графові моделі, варіаційні алгоритми, інформаційні технології, програмна інженерія, математичне моделювання, обчислювальні методи, системний аналіз.

ABSTRACT

Sapozhnyk, D. O. Methods and algorithms for solving the maximum cut problem using quantum hybrid algorithms. – Qualifying scientific work on manuscript rights.

Dissertation for obtaining the scientific degree of Doctor of Philosophy in the field of knowledge 12 Information Technologies, specialty 121 «Software Engineering». – State University «Zhytomyr Polytechnic», Ministry of Education and Science of Ukraine, Zhytomyr, 2026.

The Max-Cut graph problem belongs to the class of NP-hard combinatorial optimization problems and occupies an important place in algorithm theory, complexity theory, and applied mathematics. The purpose of the problem is to find such a division of the set of vertices of an undirected graph into two subsets, in which the number or total weight of edges between these subsets is maximum. Despite the simplicity of the formulation, the problem is characterized by an exponential growth of the space of possible solutions with an increase in the number of vertices, which determines its high computational complexity.

The Max-Cut problem is used in the tasks of optimizing the topology of telecommunication and energy networks, load balancing in distributed systems, data clustering, analysis of network structures, financial modeling, as well as in the tasks of designing integrated circuits and transport planning.

Classical approaches to the Max-Cut solution show significant constraints as the dimension of the graph increases. Precise algorithms, in particular the full traversal method and the branch and boundary method, provide for finding a globally optimal solution, however their exponential time complexity limits their use for medium - and large-dimensional problems. Approximation methods, in particular the Goemans-Williamson algorithm, provide theoretical guarantees of the quality of the solution, but are associated with significant computational costs caused by the use of semi-defined programming. Heuristic and metaheuristic algorithms such as local search, genetic algorithms and simulated annealing have better scalability,

however do not guarantee the achievement of a global optimum and are characterized by variability in results.

In the context of the development of quantum computing, the problem of finding efficient methods for solving NP-hard problems takes on a new meaning. The emergence of variational quantum algorithms, in particular Quantum Approximate Optimization Algorithm (QAOA), created prerequisites for the use of quantum resources in combinatorial optimization problems. At the same time, the current state of quantum technologies is determined by the characteristics of the NISQ-era (Noisy Intermediate-Scale Quantum), which is limited by a small number of qubits, limited by the depth of quantum circuits, as well as the influence of noise and decoherence. In such conditions, the implementation of complex parameterized quantum circuits and multi-iterative hybrid algorithms is complicated.

An additional limitation of variational quantum algorithms is the need for multiple interactions between quantum and classical components to optimize quantum circuit parameters. Such an iterative procedure is accompanied by overhead costs, increased sensitivity to noise and the complexity of the optimization landscape, which can negatively affect the efficiency of their application in the conditions of real quantum devices.

In this regard, the relevance of the thesis research topic is due to the need to develop quantum-hybrid algorithms for solving the maximum graph section problem, focused on reducing the depth of quantum circuits, reducing the number of iterations of classical optimization and increasing noise resistance in the conditions of modern quantum computing systems.

The aim of the dissertation research is the development of methods and algorithms for solving the maximum section problem using quantum hybrid algorithms.

The content of the dissertation consists of three sections, in which the main results of the conducted research are given and substantiated.

The introduction substantiates the relevance of the research topic, defines the object as well as the subject, the goal, and the tasks; it also formulates the

scientific novelty and practical significance of the obtained results. Furthermore, it presents the development methods that were used to obtain the results and indicates information about the publication and approbation of the dissertation research results.

In the first chapter, a thorough analysis of the current state of development of quantum technologies was carried out, namely, an analysis of publications over the past 10 years was carried out in the scientific metric database Scopus, which showed a significant increase in interest in quantum technologies in the field of computer science. In the area of investment, a significant increase in funding is also noticeable, which correlates with the increased interest of scientists, which is expressed through the publication of scientific works. The activities of leading companies conducting research in the field of quantum technologies were analyzed, the main directions of their development and approaches to the implementation of quantum computing systems were determined.

The topological features of the physical realization of qubits were studied and the main limitations arising in the development of quantum computers were determined. A formal description of the maximum cut problem is given and an analysis of classical and quantum-hybrid algorithms for its solution is performed. Particular attention is paid to the study of single- and multi-qubit quantum gates, which is a necessary basis for building quantum-hybrid algorithms.

The second section dedicated to the development of the quantum hybrid algorithm QASPA. Within the scope of the section, an analysis of the multi-iteration scheme of the QAOA algorithm was performed and a one-iteration approach was proposed, which allows to reduce the number of accesses to the quantum component of the algorithm. A model of the interaction of the main components of the QASPA algorithm was developed and the functional structure of each of them was determined. The classical part of the algorithm implements the formalization of the problem of the maximum section in the form of a matrix of angles, which is used to parameterize the quantum circuit.

A study of the selection of parameters was carried out, based on the results of which it was established that the use of the angle value $\pi/4$ provides stable results when solving the problem. A method for scaling the number of runs of a quantum circuit is proposed, which allows determining a sufficient number of measurements to isolate the most likely solutions against the background of noise. A method of building a quantum circuit based on the use of the Hadamard gate layer and the CX-RY-CX sequence of operations, which provide coding of the graph structure in the quantum state, has been developed. A mathematical analysis of the proposed scheme was carried out, which confirms the correctness of its construction. The final stage of the algorithm is also described, which involves analyzing the results of quantum measurements and determining the optimal solution.

In the third chapter of the dissertation research analyzed modern platforms for the development of quantum algorithms and implemented the QASPA algorithm based on the Qiskit platform. Experimental launches of the algorithm were carried out on quantum simulators and real quantum devices, which confirmed the correctness of its implementation.

To evaluate the quality of the obtained results, a series of experiments was carried out on test graphs of small and medium dimension, as well as on automatically generated graphs with the number of vertices from 3 to 14 different topologies, in particular complete, cyclic and random graphs of the Erdős-Rényi model with different connectivity coefficients. Based on the results of the experiments, it was established that the QASPA algorithm provides optimal or close to optimal solutions with an average approximation coefficient within 0.98-1.00, while in all cases the value of the coefficient exceeded the threshold value $p > 0.9$. Reference values were obtained using the full enumeration method. In addition, a comparative analysis with the QAOA algorithm was performed, the results of which demonstrate greater variability and a decrease in the quality of solutions while increasing the dimensionality of the problem.

The QASPA algorithm is also evaluated by a success rate that was 100% for all test cases regardless of the type of graph and its dimension in the studied range.

Separately, an experimental study of the time characteristics of the algorithm on complete graphs with the number of vertices up to 19 was conducted. It was established that the execution time of the algorithm is within 0.0846-0.4643 seconds and does not demonstrate exponential growth with increasing dimensionality of the problem, which indicates the stability of its computational characteristics in the studied range.

As a result of scientific research carried out within the framework of the dissertation research **for the first time** the following were received **scientific results**:

- developed a quantum hybrid QASPA algorithm for solving the maximum graph section problem, which unlike the QAOA algorithm, implements a single-iteration architecture without using multi-iteration classical parameter optimization, which allows to reduce computational costs and ensures stability of the results with achieving 100% success rate of the algorithm for the studied cases at an average approximation ratio in the range of 0.98-1.00;
- a deterministic method of graph-dependent parameterization of a variational quantum circuit based on mapping the structure of the graph into a matrix of rotation angles with a fixed value is proposed $\pi/4$ for existing edges, and implements phase coding of graph relationships through rotation about an axis O_y ;
- a method for scaling the number of quantum launches based on an affinity-exponential approximation has been developed, which allows determining a statistically sufficient number of measurements to identify the optimal solution as the dimensionality of the problems increases;
- a method of constructing quantum circuits for combinatorial optimization problems is proposed, which involves the use of a layer of Hadamard gates and a CX-RY-CX three-gate block to encode the edges of the graph and ensures a reduction in the depth of the quantum circuit and adaptation to the limitations of modern quantum processors.

Practical significance the results of the work consist of:

- development of software solutions for solving combinatorial optimization problems based on the QASPA quantum hybrid algorithm for optimization of logistics routes, load balancing in network systems and resource allocation problems;
- creation of application software modules for integration with quantum computing platforms (in particular IBM Quantum) for practical testing and use of the algorithm on simulators and real quantum processors;
- development of a methodical approach to the transformation of applied problems into graph models of the Max-Cut type and their solution using the QASPA quantum hybrid algorithm, which includes rules for building quantum circuits, determining parameters and evaluating results and can be used in decision support systems.

The implementation of the research results into practice is confirmed by certificates from LLC «Alliance Digital», from the company LLC «UKRAINIAN CENTER OF DUAL EDUCATION» and from the company LLC «N-IX DELIVERY» (references are attached).

According to the results of dissertation research, 11 scientific works were published, including 4 articles in specialized scientific publications of Ukraine of the category «B». In addition, 1 article was published in the materials of an international conference with indexing in the Scopus database and 6 in the materials of scientific conferences.

Keywords: quantum computing, hybrid algorithms, optimization algorithms, combinatorial optimization, maximum cut problem, NP-hard problems, graph models, variational algorithms, information technologies, software engineering, mathematical modeling, computational methods, systems analysis.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА

Наукові статті, в яких опубліковано основні наукові результати
дисертації:

**Статті у виданнях, включених до переліку наукових фахових видань
України категорії «Б»:**

1. Сапожник Д. О. Процес перетворення високорівневих квантових вентилів у базові для виконання на реальному квантовому комп'ютері. *Вісник Херсонського національного технічного університету*. 2025. Т. 2, № 2(93). С. 327–332. DOI: <https://doi.org/10.35546/kntu2078-4481.2025.2.2.40>.

2. Сапожник Д. О. Аналіз можливостей мови програмування Q# шляхом реалізації програми для генерації випадкових чисел. *Технічна інженерія*. 2024. № 1(93). С. 240–245. DOI: [https://doi.org/10.26642/ten-2024-1\(93\)-240-245](https://doi.org/10.26642/ten-2024-1(93)-240-245).

3. Сапожник Д. О. Розв'язання задачі максимального розрізу графа за допомогою квантово-гібридного амплітудно-стохастичного алгоритму. *Вісник Національного технічного університету «ХПІ». Серія: Системний аналіз, управління та інформаційні технології*. 2025. № 2 (14). С. 96–101. DOI: <https://doi.org/10.20998/2079-0023.2025.02.13>

4. Сапожник Д. О. Методи та напрями застосування квантових комп'ютерів у криптографії та логістиці. *Вчені записки ТНУ імені В.І. Вернадського. Серія: Технічні науки*. 2025. Т. 2, № 5. С. 260–265. DOI: <https://doi.org/10.32782/2663-5941/2025.5.2/34>

Публікації в зарубіжних періодичних фахових виданнях:

5. Sapozhnyk D. Quantum Approximate Optimization Algorithm for the Max-Cut Problem: JavaScript Programming Language Implementation. *Proceedings of International Conference on Applied Innovation in IT*. 2024. Vol. 12, iss. 2. Pp. 27–34. DOI: <https://doi.org/10.25673/118110>. (включений до міжнародної наукометричної бази Scopus).

Публікації у матеріалах наукових конференцій:

6. Сапожник Д. О. Квантове програмування мовою Q#: перспективи та застосування. *Тези XIV Міжнародної науково-технічної конференції*

«Інформаційно-комп'ютерні технології – 2024». Житомир. 28–29 берез. 2024. С. 104–105.

7. Сапожник Д. О. Квантовий гейт Адамара, реалізація у мові програмування Q#. *Тези Всеукраїнської науково-практичної on-line конференції здобувачів вищої освіти і молодих учених, присвяченої Дню науки.* Житомир. 13–17 трав. 2024. С. 50.

8. Сапожник Д. О. Перспективи розв'язання пр-повних задач за допомогою квантового програмування. *Тези XIV міжнародної наукової конференції студентів та молодих вчених «Сучасні інформаційні технології 2024»*, Одеса. 24 трав. 2024. С. 98–100.

9. Сапожник Д. О. Програмні засоби поширення публічної бібліотеки для роботи з квантовими обчисленнями у мові програмування javascript. *Проблеми інформатики та моделювання (ПІМ-2024) тези двадцять четвертої міжнародної науково-технічної конференції.* Харків. 20–23 верес. 2024. С. 123–124.

10. Сапожник Д. О., Морозов А. В. Економічні та промислові застосування квантових комп'ютерів. *Сучасні інформаційні системи та технології: матеріали VII Всеукр. наук. практ. інтернет-конф. за тематикою «Сучасні комп'ютерні системи та мережі в управлінні» (29 листопада 2024 р., м. Херсон, м. Хмельницький) / за ред. А. А. Григорової.* Херсон: Книжкове видавництво ФОП Вишемирський В. 2024. С.113-116.

11. Сапожник Д. О. QUBO матриці для розв'язання пр-повних задач на квантовому комп'ютері. *Тези доповідей VII Всеукраїнської науково-технічної конференції Комп'ютерні технології: інновації, проблеми, рішення.* Житомир. 2–3 груд. 2024. С. 242–243.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ УМОВНИХ ПОЗНАЧЕНЬ	18
ВСТУП	19
РОЗДІЛ 1. АНАЛІЗ СУЧАСНОГО СТАНУ КВАНТОВИХ ТЕХНОЛОГІЙ ДЛЯ РОЗВ’ЯЗАННЯ ЗАДАЧІ МАХ-CUT	26
1.1. Сучасний стан розвитку квантових технологій	26
1.2. Класичні алгоритми для розв’язання Мах-Cut.....	36
1.3. Квантові та гібридні підходи до вирішення Мах-Cut.....	53
Висновки до першого розділу.....	73
РОЗДІЛ 2. МЕТОДИ ТА МОДЕЛІ РОЗРОБКИ КВАНТОВО- ГІБРИДНОГО АЛГОРИТМУ	76
2.1. Моделювання квантово-гібридного алгоритму QASPA	76
2.2. Методи генерації кутів та масштабування кількості запусків алгоритму QASPA	88
2.3. Метод побудови квантової схеми алгоритму QASPA.....	99
Висновки до розділу 2	112
РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ QASPA	114
3.1. Програмна реалізація QASPA	114
3.2. Запуск алгоритму на квантовому симуляторі та на реальному квантовому пристрої	123
3.3. Тестування та порівняння з іншими алгоритмами для розв’язання Мах-Cut.....	142
Висновки до розділу 3	161
ЗАГАЛЬНІ ВИСНОВКИ	164
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	168
ДОДАТКИ	185
Додаток А.....	185
Додаток Б	226
Додаток В.....	228

Додаток Г	230
Додаток Д.....	233

ПЕРЕЛІК ПРИЙНЯТИХ УМОВНИХ ПОЗНАЧЕНЬ

Max-Cut –	задача максимального розрізу (Maximum Cut Problem)
GW –	алгоритм Гоєманса–Вільямсона (Goemans-Williamson algorithm)
VQA –	варіаційний квантовий алгоритм (Variational Quantum Algorithm)
QAOA –	квантовий алгоритм приблизної оптимізації (Quantum Approximate Optimization Algorithm)
NISQ –	квантові пристрої з обмеженими ресурсами (Noisy Intermediate-Scale Quantum)
QASPA –	квантово-гібридний амплітудно-стохастичний алгоритм (Quantum Approximate Shift-Phase Algorithm)
KO –	комбінаторна оптимізація
ILP –	цілочисельне лінійне програмування (Integer Linear Programming)
ZNE –	Екстраполяція нульового шуму (zero-noise extrapolation)
DD –	Стратегія динамічного розчеплення (dynamical decoupling)

ВСТУП

Актуальність дослідження. Задача максимального розрізу (Max-Cut problem) є однією з фундаментальних задач комбінаторної оптимізації, що належить до класу NP-складних проблем. Вона знаходить застосування в аналізі мережових структур, кластеризації даних, проєктуванні енергетичних систем, задачах статистичної фізики та оптимізації транспортних процесів. Незважаючи на значний обсяг досліджень, ефективні методи отримання точних розв'язків для задач великої розмірності відсутні, що обумовлює використання наближених та евристичних підходів.

Класичні методи розв'язання задачі Max-Cut, включно з алгоритмом Гоєманса-Вільямсона (Goemans-Williamson), жадібними, еволюційними алгоритмами та методами локального пошуку, забезпечують обмежену якість розв'язків або супроводжуються значними обчислювальними витратами. Зокрема, апроксимаційні алгоритми не гарантують отримання точного розв'язку, тоді як застосування методів напіввизначеного програмування для великих графів є обчислювально складним. Евристичні підходи, хоча й мають кращу масштабованість, але не забезпечують формальних гарантій якості розв'язків. Це зумовлює необхідність пошуку альтернативних обчислювальних підходів.

Одним із напрямів розвитку є застосування квантових обчислень, зокрема варіаційних квантових алгоритмів, що поєднують квантові та класичні обчислювальні ресурси. Алгоритми типу Quantum Approximate Optimization Algorithm (QAOA) демонструють можливість отримання наближених розв'язків задач оптимізації, однак їх ефективність обмежується складністю параметричного простору, залежністю від початкових умов та впливом шуму.

Сучасний етап розвитку квантових обчислень, що характеризується як NISQ-ера, накладає додаткові обмеження, пов'язані з обмеженою кількістю кубітів, шумом, похибками гейтів та топологією з'єднань. У таких умовах актуальним є розроблення квантово-гібридних алгоритмів, адаптованих до реальних характеристик квантових пристроїв та здатних забезпечувати

стабільні результати без значних витрат обчислювальних ресурсів.

У цьому контексті актуальним є дослідження методів побудови квантово-гібридних алгоритмів для задачі Max-Cut, зокрема підходів, що зменшують залежність від ітераційної класичної оптимізації та враховують структурні властивості графа при формуванні квантової схеми. Це дозволяє підвищити ефективність обчислювального процесу та адаптувати алгоритми до умов сучасних квантових обчислювальних систем.

Таким чином, обрана тематика дослідження є актуальною, оскільки спрямована на розроблення методів квантово-гібридного розв'язання задач комбінаторної оптимізації з урахуванням обмежень сучасних квантових технологій.

Зв'язок роботи з науковими програмами, планами та темами. Тема дисертації затверджена Вченою радою Державного університету «Житомирська політехніка» (протокол № 15 від 29.08.2025 р.)

Метою дисертаційного дослідження є розробка методів та алгоритмів для розв'язання задачі максимального розрізу за допомогою квантово-гібридних алгоритмів.

Визначена мета досягається шляхом вирішення таких **завдань**:

- провести аналіз сучасного стану квантових технологій для розв'язання задачі Max-Cut;
- розробити квантово-гібридний алгоритм для розв'язання задачі максимального розрізу графа, запропонувавши детермінований метод графо-залежної параметризації варіаційної квантової схеми;
- розробити метод масштабування кількості квантових запусків на основі афінно-експоненційної апроксимації та удосконалити метод побудови квантових схем для задач комбінаторної оптимізації;
- розробити програмні рішення для розв'язання задачі максимального розрізу графа та створити прикладні програмні модулі для інтеграції з платформами квантових обчислень (зокрема IBM Quantum), провести

експерименти запуску розробленого алгоритму на симуляторах та реальних квантових пристроях;

– провести порівняльний аналіз запропонованого алгоритму з існуючими квантово-гібридними методами розв’язання задачі Max-Cut та оцінити його ефективність за ключовими критеріями.

Об’єкт дослідження – процес розв’язання задачі максимального розрізу на графах різної складності.

Предметом дослідження є методи та алгоритми квантово-гібридної оптимізації орієнтовані на розв’язання задачі максимального розрізу за допомогою квантово-гібридних алгоритмів.

У дисертаційному дослідженні використовувався комплекс **методів**, які охоплюють теоретичні, алгоритмічні, програмно-технічні та експериментальні аспекти. *Теоретичний аналіз та огляд літератури* застосовувався для критичного вивчення існуючих класичних та квантових підходів до розв’язання задачі Max-Cut, виявлення їхніх переваг та обмежень, а також для обґрунтування наукової та практичної доцільності розробки нового гібридного алгоритму в умовах Noise-Intermediate Scale Quantum ери. Під час аналізу предметної області комбінаторної оптимізації, формалізації постановки задачі Max-Cut у контексті NP-складних задач, побудови структурної моделі взаємодії класичних та квантових компонентів у рамках розробленого алгоритму QASPA було використано *системний підхід*. *Математичне моделювання* застосовувалося для формалізації задачі Max-Cut, розробки та обґрунтування моделі приведення графа до матриці кутів (включаючи нормування та вибір фіксованого кута $\pi/4$), а також для побудови афінно-експоненційної апроксимації для визначення оптимальної кількості квантових запусків. *Методи теорії алгоритмів* використовувалися для розробки та формалізації нового квантово-гібридного алгоритму QASPA, його архітектури (одноітераційна модель, побудова квантової схеми за допомогою гейтів Адамара та керованих обертань навколо осі O_y), а також для аналізу часової та просторової складності, порівняння з існуючими класичними та

квантовими алгоритмами. *Об'єктно-орієнтоване моделювання і програмування* застосовувалося для проєктування архітектури програмного забезпечення, структурування модулів та компонентів алгоритму QASPA, а також для реалізації класичних та квантових модулів із використанням принципів об'єктно-орієнтованої парадигми. *Моделювання квантових обчислень* здійснювалося за допомогою фреймворку Qiskit для побудови квантових схем QASPA, симуляції їх виконання на Qiskit AerSimulator, а також для підготовки та запуску алгоритмів на реальних квантових процесорах IBM Quantum, включаючи застосування технік транспіляції та пом'якшення помилок. Методи експериментальних досліджень використовувалися для апробації роботи розробленого алгоритму QASPA у симуляційних середовищах та на реальних квантових процесорах, оцінки його стійкості до шуму, перевірки ефективності на графах різної розмірності та порівняння результатів із класичними підходами та QAOA. *Методи порівняльного аналізу* застосовувалися для систематичного тестування та зіставлення продуктивності QASPA з методом повного перебору та стандартною реалізацією QAOA на шести наборах графів: довільних графах малої розмірності, повних, циклічних та випадкових графах з різним коефіцієнтом щільності. *Методи статистичного аналізу* використовувалися для оцінки якості рішень, аналізу ймовірнісних розподілів результатів вимірювань на квантових схемах, а також для верифікації шумостійкості алгоритму та обґрунтування моделі масштабування кількості запусків. *Методи аналізу складності* застосовувалися для визначення часових характеристик запропонованого алгоритму QASPA, порівняння ефективності з існуючими підходами та визначення його потенційних меж масштабованості в контексті зростання розміру графа.

Використання технологій штучного інтелекту. У межах виконання дисертаційного дослідження технології штучного інтелекту застосовувалися виключно як допоміжний засіб візуалізації з метою більш наочного представлення окремих розроблених схем, створених на основі авторського

опису.

Усі наукові положення, а також мета і завдання дослідження сформульовані автором особисто; експериментальні дослідження виконано самостійно, а обробку отриманих результатів і формування висновків здійснено без сторонньої допомоги. Технології штучного інтелекту не використовувалися для створення тексту дисертації або формулювання наукових результатів.

Наукова новизна та теоретичне значення отриманих результатів полягають у тому, що **вперше**:

- розроблено квантово-гібридний алгоритм QASPA для розв’язання задачі максимального розрізу графа, який, на відміну від алгоритму QAOA, реалізує одноітераційну архітектуру без використання багатоітераційної класичної оптимізації параметрів, що дозволяє зменшити обчислювальні витрати та забезпечує стабільність результатів із досягненням 100% успішності виконання алгоритму для досліджених випадків при середньому коефіцієнті апроксимації в межах 0.98-1.00;

- запропоновано детермінований метод графо-залежної параметризації варіаційної квантової схеми, що базується на відображенні структури графа у матрицю кутів обертання з фіксованим значенням $\pi/4$ для наявних ребер та реалізує фазове кодування зв’язків графа через обертання навколо осі O_y ;

- розроблено метод масштабування кількості квантових запусків на основі афінно-експоненційної апроксимації, який дозволяє визначати статистично достатню кількість вимірювань для ідентифікації оптимального розв’язку при зростанні розмірності задач;

- запропоновано метод побудови квантових схем для задач комбінаторної оптимізації, що передбачає використання шару гейтів Адамара та тригейтового блоку CX-RY-CX для кодування ребер графа та забезпечує зменшення глибини квантового кола і адаптацію до обмежень сучасних квантових процесорів.

Практичне значення результатів роботи полягає у:

- розробці програмних рішень для розв’язання задач комбінаторної оптимізації на основі квантово-гібридного алгоритму QASPA для оптимізації логістичних маршрутів, балансування навантаження в мережевих системах та задачах розподілу ресурсів;

- створенні прикладних програмних модулів для інтеграції з платформами квантових обчислень (зокрема IBM Quantum) для практичної апробації та використання алгоритму на симуляторах і реальних квантових процесорах;

- розробці методичного підходу до перетворення прикладних задач у графові моделі типу Max-Cut та їх розв’язання за допомогою квантово-гібридного алгоритму QASPA, що включає правила побудови квантових схем, визначення параметрів та оцінювання результатів і може бути використаний у системах підтримки прийняття рішень.

Особистий внесок здобувача. Усі основні результати дисертаційного дослідження, які представлені до захисту, одержані автором особисто (звіт подібності сформований за допомогою програми StrikePlagiarism додається).

У працях, опублікованих у співавторстві, особистий внесок здобувача полягає у проведенні аналізу сучасного стану розвитку квантових обчислень та їх застосувань в економіці й промисловості, узагальненні наукових джерел з питань квантового відпалу та універсальних квантових обчислень, дослідженні можливостей використання квантових технологій у фінансах, логістиці, енергетиці, хімічній промисловості та машинному навчанні [3].

Апробація результатів дисертації. Основні результати наукових досліджень неодноразово доповідалися на ряді міжнародних та Всеукраїнських наукових конференціях та семінарах, зокрема: XIV Міжнародній науково-технічній конференції «Інформаційно-комп’ютерні технології – 2024» (28–29 березня 2024 р., м. Житомир), Всеукраїнській науково-практичній on-line конференції здобувачів вищої освіти і молодих учених, присвяченій Дню науки (13–17 травня 2024 р., м. Житомир), XIV Міжнародній науковій конференції студентів та молодих учених «Сучасні

інформаційні технології – 2024» (24 травня 2024 р., м. Одеса), XXIV Міжнародній науково-технічній конференції «Проблеми інформатики та моделювання (ПІМ–2024)» (20–23 вересня 2024 р., м. Харків), Всеукраїнській науково-технічній конференції «Комп’ютерні технології: інновації, проблеми, рішення» (2–3 грудня 2024 р.), Всеукраїнській науково-практичній інтернет-конференції студентів, аспірантів та молодих учених (29 листопада 2024 р., м. Хмельницький), VII Всеукраїнської науково-технічної конференції Комп’ютерні технології: інновації, проблеми, рішення (2–3 груд. 2024, м. Житомир).

Впровадження результатів дослідження в практику підтверджується довідками у ТОВ «Альянс Діджитал», у компанії ТОВ «УКРАЇНСЬКИЙ ЦЕНТР ДУАЛЬНОЇ ОСВІТИ» та у компанії ТОВ «Н-ІКС ДЕЛІВЕРІ» (довідки додаються).

Публікації. За результатами дисертаційних досліджень було опубліковано 11 наукових праць, з них 4 статті у фахових наукових виданнях України категорії «Б», 1 стаття опублікована у матеріалах міжнародної конференції з індексацією у базі даних Scopus та 6 у матеріалах наукових конференцій.

Структура та обсяг дисертації. Дисертаційна робота складається з анотації, вступу, трьох розділів, висновків до кожного розділу, висновків, списку використаних джерел (136 найменувань, з них 131 іноземною мовою), 5 додатків. Загальний обсяг дисертації – 232 сторінки, з них 165 сторінок основного тексту, у якому міститься 2 таблиці та 63 рисунки.

РОЗДІЛ 1. АНАЛІЗ СУЧАСНОГО СТАНУ КВАНТОВИХ ТЕХНОЛОГІЙ ДЛЯ РОЗВ’ЯЗАННЯ ЗАДАЧІ MAX-CUT

1.1. Сучасний стан розвитку квантових технологій

Еволюція квантових обчислень бере свій початок у 1980-х роках, зокрема з робіт Фейнмана Р. (Feynman R.), у яких було обґрунтовано доцільність використання квантових систем для моделювання квантових процесів [22]. Ці ідеї стали підґрунтям для формування квантової обчислювальної парадигми. Подальший розвиток даного напрямку у 1990-х роках пов’язаний із появою ключових квантових алгоритмів, зокрема алгоритму Шора та алгоритму Гровера, які продемонстрували можливість прискорення обчислень порівняно з класичними підходами для окремих класів задач [96]. Алгоритм Шора, запропонований Шором П. (Shor P.), забезпечує експоненційне прискорення задачі факторизації цілих чисел [96]. У свою чергу, алгоритм Гровера, розроблений Гровером Л. (Grover L.), забезпечує квадратичне прискорення пошуку в неструктурованих даних [37].

Основні етапи розвитку квантових обчислень охоплюють період від появи перших експериментальних квантових комп’ютерів, як от системи IBM, представленої у 1998-2001 роках, що ознаменували перехід від теоретичних концепцій у початок практичної реалізації [135], до сучасного стану, коли провідні технологічні компанії (рис. 1.1), включаючи IBM, Intel, Google та Microsoft, активно розробляють та вдосконалюють квантові системи. Ці компанії не лише інвестують у створення потужніших квантових процесорів, але й надають хмарний доступ до своїх квантових ресурсів, що сприяє подальшим дослідженням, розробкам та ширшому впровадженню у цій галузі.

IBM є однією з компаній, що здійснюють системні дослідження у галузі квантових обчислень та розробку надпровідних квантових процесорів на основі трансмонних кубітів. У межах розвитку апаратного забезпечення компанією представлено низку процесорів із поступовим збільшенням кількості кубітів, зокрема у 2023 році – процесор Condor із 1121 кубітом, а

також 133-кубітний процесор Heron [10]. Зазначені розробки супроводжуються дослідженнями, спрямованими на зменшення рівня квантових помилок та підвищення стабільності обчислень, що є важливим для практичного використання квантових систем [98]. Паралельно розвивається підхід до побудови масштабованих обчислювальних систем, який передбачає інтеграцію квантових та класичних ресурсів у межах гібридних обчислювальних архітектур [7].

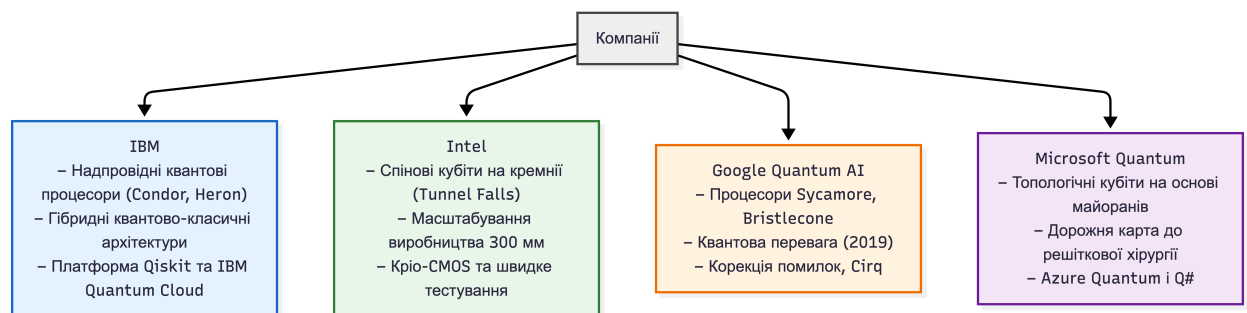


Рис. 1.1. Компанії які проводять квантові дослідження

Відповідно до оприлюднених планів розвитку, у рамках платформи IBM Quantum передбачається подальше вдосконалення квантових процесорів, зокрема створення систем із більшою кількістю кубітів та покращеними характеристиками виконання квантових операцій [7]. Окрім апаратного забезпечення, IBM розробляє програмні засоби для роботи з квантовими системами. Одним із таких інструментів є Qiskit – відкрита програмна платформа, що надає можливість розробки, моделювання та виконання квантових алгоритмів із використанням як симуляторів, так і реальних квантових пристроїв через хмарну інфраструктуру [87]. У межах досліджень також розглядаються методи зменшення впливу шуму, зокрема імовірнісна відміна помилок та екстраполяція нульового шуму, які застосовуються для підвищення точності результатів на сучасних квантових пристроях [7]. Активність використання квантових систем IBM підтверджується значною кількістю наукових публікацій, у яких застосовуються відповідні обчислювальні ресурси, а також великою кількістю виконаних квантових схем у межах хмарної платформи IBM Quantum [17].

Intel у сфері квантових обчислень зосереджується на розробці платформ на основі спінових кубітів, використовуючи підходи, сумісні з напівпровідниковим виробництвом, зокрема інфраструктуру 300-міліметрових технологічних ліній [73]. Такий підхід передбачає можливість інтеграції квантових пристроїв у наявні виробничі процеси. У межах досліджень компанією реалізовано прототипи лінійних масивів спінових кубітів, зокрема систему «Tunnel Falls», що включає 12 квантових точок [73]. Також продемонстровано виготовлення кремнієвих спінових кубітів на 300-міліметрових кремнієвих пластинах, що вказує на сумісність таких підходів із сучасними технологіями мікроелектронного виробництва [49]. Окремим напрямом досліджень є розробка методів тестування квантових пристроїв, зокрема із застосуванням кріогенних вимірювальних систем, що дозволяють скоротити час перевірки параметрів кубітів [49]. Крім того, досліджується інтеграція керуючої електроніки безпосередньо в кріогенні середовища, що спрямовано на зменшення кількості зовнішніх з'єднань та підвищення масштабованості систем, зокрема, продемонстровано керування спіновими кубітами із використанням кріогенного CMOS-чипа, що функціонує при наднизьких температурах [52]. Подальші дослідження Intel спрямовані на збільшення кількості кубітів у межах одного чипа з урахуванням сумісності спінових кубітів із технологіями кремнієвого виробництва [125].

Google Quantum AI здійснює дослідження у сфері квантових обчислень, зокрема у напрямі розробки надпровідних квантових процесорів. У 2019 році компанією було продемонстровано виконання обчислювальної задачі на процесорі Sycamore із 53 кубітами [8]. Згідно з опублікованими результатами, час виконання цієї задачі становив близько 200 секунд, тоді як її моделювання на класичних обчислювальних системах оцінювалося як значно більш ресурсоємне [8]. Подальші дослідження пов'язані з розвитком квантових процесорів, зокрема Bristlecone, із фокусом на збільшенні кількості кубітів, підвищенні часу когерентності та зменшенні рівня помилок [8]. Значна увага також приділяється методам квантової корекції помилок, що є необхідними

для підвищення надійності квантових обчислень [49]. У межах цих досліджень продемонстровано зниження рівня помилок із використанням циклічних схем корекції, а також досягнення режимів роботи, що відповідають умовам застосування поверхневих кодів корекції помилок [75]. Окрім цього, квантові процесори використовуються для дослідження фізичних ефектів, зокрема для спостереження часових кристалів та крайових мод Майорани [51]. У програмному забезпеченні Google Quantum AI розвиває фреймворк Cirq, який використовується для побудови та моделювання квантових схем і орієнтований на роботу з шумними квантовими пристроями проміжного масштабу [23].

Microsoft здійснює дослідження у сфері квантових обчислень, зокрема у напрямі розробки топологічних кубітів, які розглядаються як один із підходів до підвищення стійкості квантових систем до помилок [6]. У межах цього підходу досліджуються фізичні реалізації кубітів на основі майоранівських ферміонів, які, згідно з теоретичними моделями, можуть характеризуватися підвищеною стійкістю до декогерентності. У наукових роботах повідомляється про спостереження топологічного зазору у гетероструктурах індію арсеніду-алюмінію, що розглядається як один із кроків у напрямі створення відповідних квантових елементів [49].

Подальші дослідження компанії спрямовані на реалізацію архітектур, що підтримують операції над логічними кубітами із застосуванням методів квантової корекції помилок [46]. Паралельно Microsoft розвиває хмарну платформу Azure Quantum, яка забезпечує доступ до інструментів для розробки, моделювання та виконання квантових програм із використанням гібридних обчислювальних підходів [103]. Одним із компонентів цієї платформи є Quantum Development Kit, що включає мову програмування Q#, призначену для реалізації квантових алгоритмів, а також підтримує інтеграцію з іншими квантовими інструментами та мовами програмування [4].

Діяльність IBM, Intel, Google Quantum AI та Microsoft охоплює різні напрями розвитку квантових технологій, включаючи апаратні рішення,

програмні платформи та алгоритмічні підходи. Сукупність цих досліджень формує сучасний стан розвитку квантових обчислень та визначає передумови для подальших досліджень у цій галузі.

Згідно з представленими даними (рис. 1.2), у сфері квантових технологій спостерігається зростання обсягів інвестицій з боку державного та приватного секторів. У 2014-2016 роках обсяги фінансування перебували на рівні 0,1-0,2 млрд доларів США, тоді як починаючи з 2017 року фіксується поступове збільшення інвестицій у квантові дослідження та розробки. Подальше зростання обсягів фінансування після 2020 року у наукових джерелах пов'язується з розширенням досліджень у галузі квантових обчислень, розвитком хмарних квантових платформ та участю великих технологічних компаній у відповідних проєктах [128].

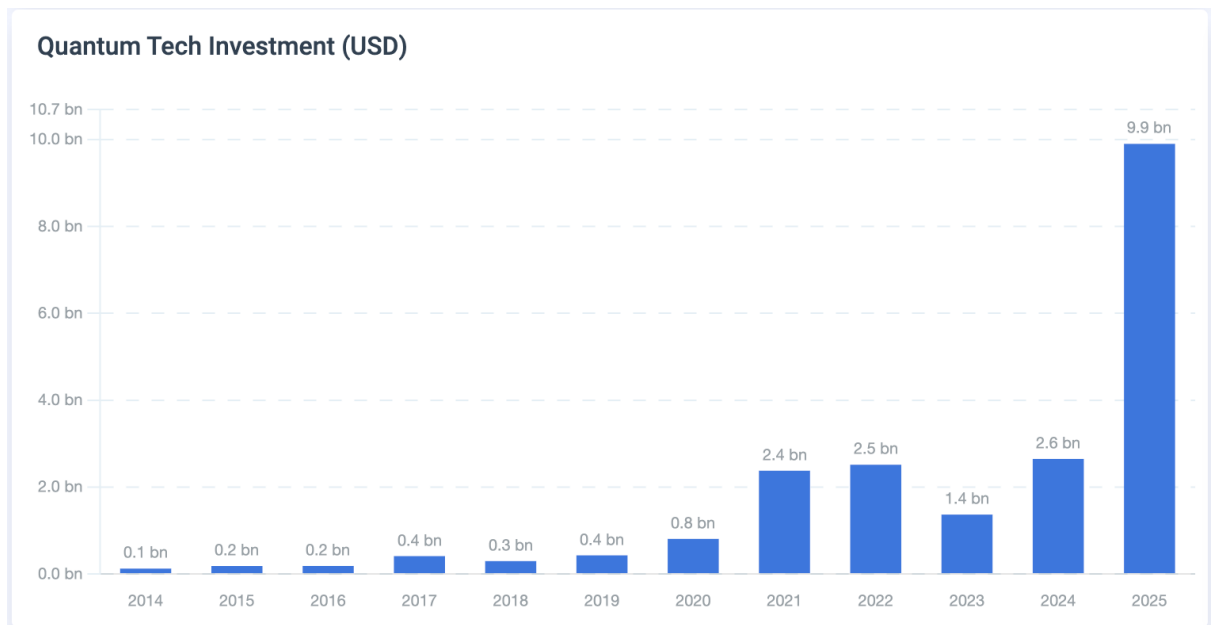


Рис. 1.2. Динаміка інвестицій у квантові технології [128]

У період 2021–2022 років обсяг інвестицій перевищив 2,4-2,5 млрд доларів США, що відображає збільшення фінансової активності у даній сфері. У 2023 році спостерігалось певне зниження фінансування, після чого у 2024 році знову зафіксовано зростання, а у 2025 році обсяг інвестицій наблизився до 10 млрд доларів США, що є максимальним значенням за наведений період спостережень. Така динаміка може бути пов'язана із розширенням сфер

застосування квантових технологій, а також із підвищенням інтересу до задач оптимізації, моделювання та аналізу даних [114].

Зростання інвестицій також розглядається у контексті розвитку сучасних інформаційних технологій, де квантові обчислення можуть застосовуватися у задачах кібербезпеки, зокрема у зв'язку з аналізом стійкості криптографічних протоколів, а також у задачах штучного інтелекту, оптимізації та моделювання складних систем [74]. Аналіз динаміки кількості наукових публікацій за ключовим словом quantum (рис. 1.3), проіндексованих у базі даних Scopus за період 2015-2025 років, демонструє зростання кількості робіт у галузі Computer Science. У 2015-2019 роках кількість публікацій залишалася відносно стабільною та коливалася в межах 650-750 робіт на рік, що відображає сталий рівень наукової активності у відповідний період.

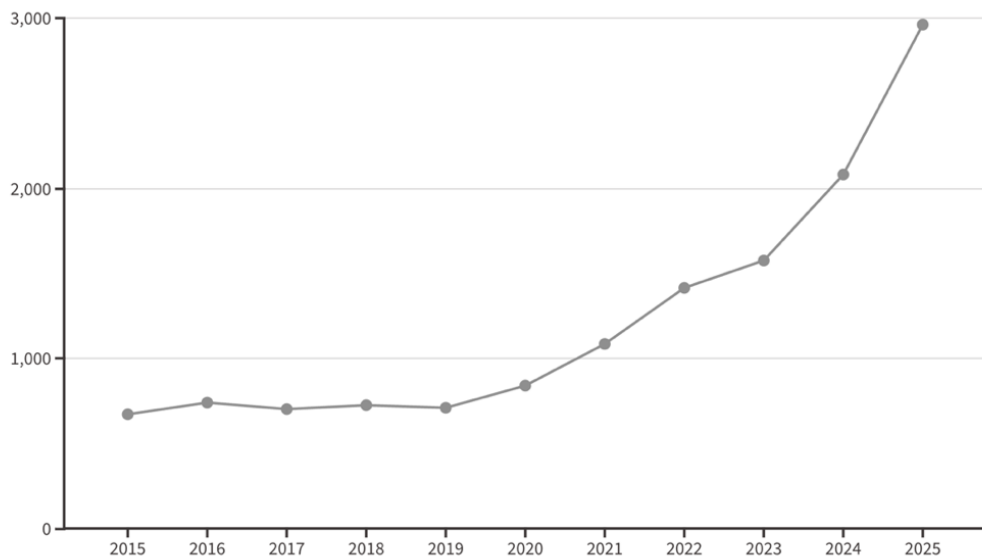


Рис. 1.3. Кількість наукових публікацій проіндексованих у Scopus за 10 років у сфері Computer Science за ключовим словом quantum [113]

Починаючи з 2020 року спостерігається подальше збільшення кількості публікацій: у 2020 році їх кількість перевищила 800, у 2021 році – понад 1100, у 2022-2023 роках – досягла приблизно 1400-1600 робіт на рік. У 2024-2025 роках кількість публікацій перевищила 2000 на рік, а у 2025 році наблизилася до 3000 робіт. Така динаміка відображає зростання публікаційної активності у сфері комп'ютерних наук, у напрямку квантових обчислень. Наведені

статистичні дані свідчать про розширення обсягів наукових досліджень та підвищення уваги до міждисциплінарних напрямів, що поєднують квантові обчислення з іншими галузями сучасних інформаційних технологій.

Однією з ключових концепцій у дослідженнях квантових обчислень є поняття квантової переваги, яке визначається як здатність квантових комп'ютерів виконувати обчислювальні задачі, що є практично недосяжними для класичних обчислювальних систем протягом прийнятного часу [31]. Це поняття використовується для опису відмінностей у обчислювальних можливостях квантових і класичних підходів. Дослідження, пов'язані з досягненням квантової переваги, розглядаються як один із напрямів розвитку квантових технологій, оскільки вони спрямовані на виявлення класів задач, у яких квантові алгоритми можуть демонструвати переваги над класичними методами. Водночас практична реалізація таких підходів є складною задачею, що потребує розвитку як апаратного забезпечення квантових комп'ютерів, так і відповідних алгоритмічних методів [133].

Незважаючи на активний розвиток квантових технологій, реалізація масштабованих квантових комп'ютерів пов'язана з низкою технічних обмежень, характерних для епохи шумних квантових пристроїв проміжного масштабу (NISQ), яка характеризується обмеженою кількістю кубітів, похибками квантових операцій та впливом декогерентності [96]. Такі обмеження зумовлюють розвиток гібридних квантово-класичних підходів, що поєднують використання квантових процесорів для виконання окремих обчислювальних підзадач із застосуванням класичних алгоритмів оптимізації [96]. Реалізація квантових обчислювальних систем на фізичному рівні пов'язана з використанням різних технологічних платформ, кожна з яких характеризується специфічними особливостями та обмеженнями. Серед підходів до фізичної реалізації кубітів (рис. 1.4) розглядаються іонні пастки та надпровідні кубіти, а також інші технології, зокрема квантові точки, нейтральні атоми, фотонні системи та топологічні кубіти, які перебувають на різних стадіях дослідження та розробки [45].

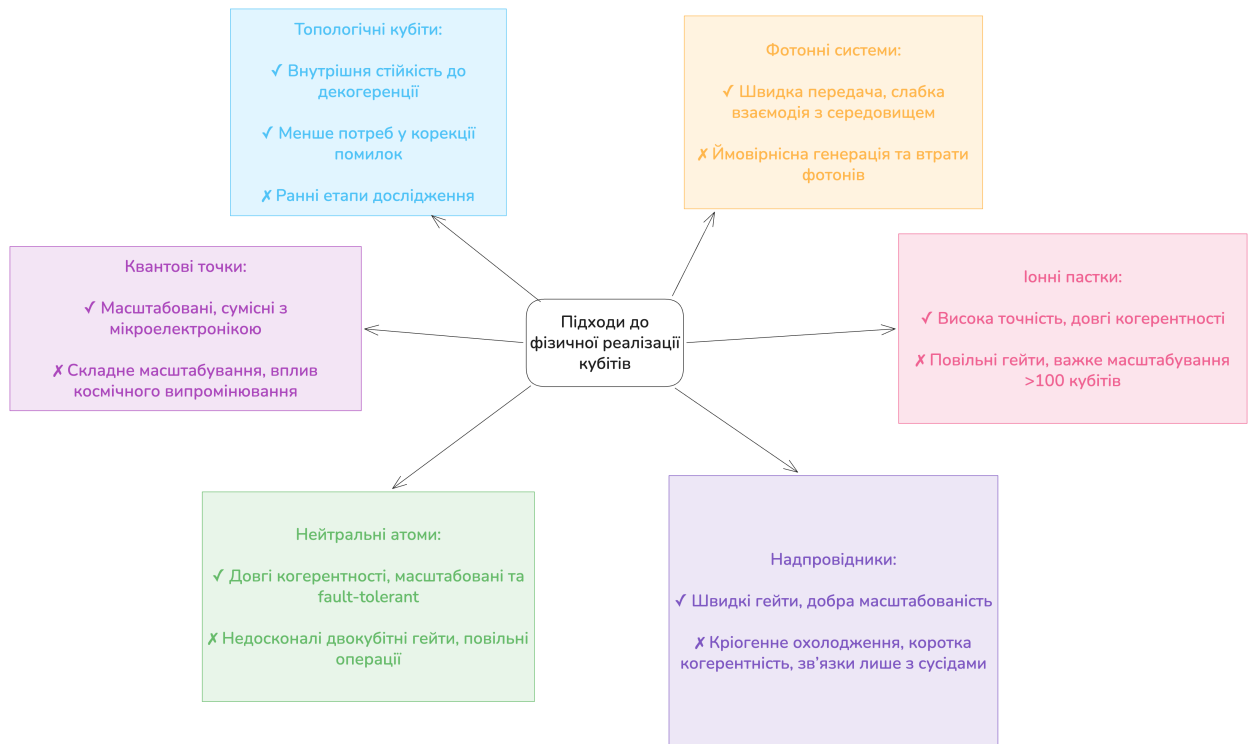


Рис. 1.4. Підходи до фізичної реалізації кубітів

Іонні пастки є одним із підходів до реалізації квантових обчислень, у якому кубіти кодуються у внутрішніх енергетичних станах окремих електрично заряджених атомів (іонів), що охолоджуються лазерним випромінюванням та утримуються за допомогою електричних полів, формуючи впорядковані структури у вакуумному середовищі [58]. До характеристик іонних пасток відносять високу точність виконання квантових операцій та тривалий час когерентності, що зумовлено відносною ізолюваністю іонів від зовнішнього середовища. Крім того, у таких системах можливе виконання операцій між довільними парами кубітів, що відповідає повнозв'язаній топології, і забезпечує реалізацію одно- та двокубітних квантових гейтів [17]. Водночас обмеженням цієї технології є відносно низька швидкість виконання квантових операцій (на рівні мікросекунд), а також складність масштабування систем при збільшенні кількості кубітів понад десятки елементів, що пов'язано з необхідністю контролю колективних рухових мод іонів [17].

Надпровідні кубіти використовують квантово-механічні властивості макроскопічних електричних ланцюгів, що формуються внаслідок утворення куперівських пар електронів у надпровідниках при наднизьких температурах [17]. Такі системи, які часто розглядаються як «штучні атоми», реалізуються у надпровідних схемах, де перехід Джозефсона забезпечує нелінійність, необхідну для формування дискретних енергетичних рівнів [15]. До характеристик надпровідних кубітів відносять високу швидкість виконання квантових операцій, що дає змогу реалізовувати значну кількість гейтів за короткий проміжок часу. Також зазначена технологія застосовується при побудові багатокубітних систем, зокрема з використанням інтегральних підходів до масштабування [17]. Водночас реалізація таких систем потребує використання кріогенних температур, що зумовлює необхідність застосування спеціалізованого охолоджувального обладнання [17]. Крім того, надпровідні кубіти зазвичай характеризуються обмеженою топологією зв'язків між сусідніми кубітами, а також меншими часами когерентності порівняно з окремими іншими технологічними підходами, це зумовлює необхідність ефективної компіляції та оптимізації квантових схем при реалізації складних алгоритмів [17].

Побудова квантових бітів на основі *нейтральних атомів* реалізується шляхом їх утримання в оптичних пастках або оптичних ґратках, тоді як маніпулювання квантовими станами здійснюється за допомогою лазерних або мікрохвильових імпульсів [53]. До характеристик такого підходу відносять тривалий час когерентності та можливість масштабування систем, що зумовлює його використання у дослідженнях багатокубітних квантових систем. Зазначені властивості також розглядаються у контексті побудови відмовостійких квантових архітектур, водночас дана технологія потребує подальшого вдосконалення точності двокубітних гейтів та підвищення швидкості виконання квантових операцій [53].

Фотонні квантові комп'ютери використовують світло як носій квантової інформації, характеризуючись можливістю швидкої передачі даних

та відносно низькою чутливістю до декогерентності, що зумовлено слабкою взаємодією фотонів із навколишнім середовищем, водночас до обмежень даного підходу відносять імовірнісний характер генерації фотонів-кубітів, а також втрати фотонів під час передачі та обробки інформації [30].

Квантові комп'ютери на основі квантових точок та *спінових кубітів* у кремнії поєднують квантові властивості з технологіями напівпровідникового виробництва, забезпечуючи сумісність із сучасною мікроелектронікою та розглядаючись у контексті побудови масштабованих квантових систем, проте масштабування таких систем до великої кількості кубітів, що перебувають у взаємодії, залишається складною задачею [73].

Топологічні кубіти, у свою чергу, базуються на використанні квазічастинок, зокрема майоранівських ферміонів, та у теоретичних моделях розглядаються як такі, що характеризуються підвищеною стійкістю до декогерентності і зменшеними вимогами до корекції помилок, проте їхня фізична реалізація залишається складною задачею і перебуває на ранніх стадіях дослідження [46].

Кожна з розглянутих технологічних платформ характеризується специфічними обмеженнями, що можуть включати вплив зовнішніх факторів, зокрема космічного випромінювання, на твердотільні кубіти, а також необхідність розробки алгоритмічних підходів, спрямованих на зменшення розмірності квантових задач у зв'язку з обмеженим розміром квантових регістрів.

Архітектурні обмеження розглядаються як один із факторів, що впливають на розвиток квантових обчислювальних систем, при цьому топологія зв'язків між кубітами, яка може бути лінійною, сітчастою або повнозв'язаною, визначає особливості виконання квантових алгоритмів та ефективність побудови квантових схем [2]. Високорівневі квантові вентиля, такі як гейти Адамара, Тоффолі та Фредкіна, у більшості випадків не реалізуються безпосередньо на сучасних квантових процесорах у зв'язку з апаратними обмеженнями, що зумовлює необхідність їх декомпозиції на

базові гейти, підтримувані конкретною апаратною платформою [2]. Сучасні квантові пристрої характеризуються обмеженою кількістю кубітів, обмеженою зв'язністю між ними та наявністю шумів різної природи, що впливають на коректність виконання квантових схем та якість отриманих результатів [12].

Інтеграція апаратного та програмного забезпечення квантових систем розглядається як міждисциплінарна задача, що охоплює рівні апаратної реалізації, програмного забезпечення, мов програмування та алгоритмічних підходів, при цьому розробка єдиного програмного стека, який забезпечує розподіл обчислювальних задач між класичними та квантовими ресурсами, а також планування використання квантових прискорювачів, розглядається як один із напрямів розвитку квантових обчислювальних платформ.

1.2. Класичні алгоритми для розв'язання Max-Cut

Комбінаторна оптимізація (КО) є розділом оптимізації, що розглядає задачі знаходження найкращої конфігурації або значення серед скінченної множини дискретних варіантів, при цьому змінні у таких задачах набувають дискретних значень, а множина можливих розв'язків є обмеженою [82]. Проблеми комбінаторної оптимізації застосовуються у широкому колі прикладних задач, зокрема у логістиці, транспортуванні, розподілі ресурсів та плануванні [94].

Задачі КО можуть бути сформульовані як задача пошуку елемента множини, що мінімізує або максимізує задану функцію витрат, до типових прикладів належать задача комівояжера, задача про рюкзак, задача максимального розрізу та задачі розкладу [85]. Багато задач комбінаторної оптимізації належать до класу NP-складних, що означає відсутність відомих алгоритмів їх розв'язання за поліноміальний час у загальному випадку, проте на практиці для таких задач застосовуються алгоритмічні підходи, здатні обробляти задачі великої розмірності за рахунок використання структури задачі або евристичних методів [54]. Теорія складності обчислень класифікує

обчислювальні задачі залежно від ресурсів, зокрема часу та пам'яті, необхідних для їх розв'язання (рис. 1.5), при цьому виділяють класи задач, серед яких базовими є класи P та NP .

Клас P включає задачі прийняття рішень, для яких існує алгоритм із поліноміальним часом виконання на детермінованій машині Тюрінга, тобто час виконання такого алгоритму зростає поліноміально відносно розміру вхідних даних [109]. Клас NP охоплює задачі прийняття рішень, для яких існує сертифікат розв'язку, правильність якого може бути перевірена детермінованим алгоритмом за поліноміальний час [109], при цьому клас P є підмножиною класу NP , оскільки для будь-якої задачі, що допускає розв'язання за поліноміальний час, відповідний розв'язок також може бути перевірений за поліноміальний час [95].

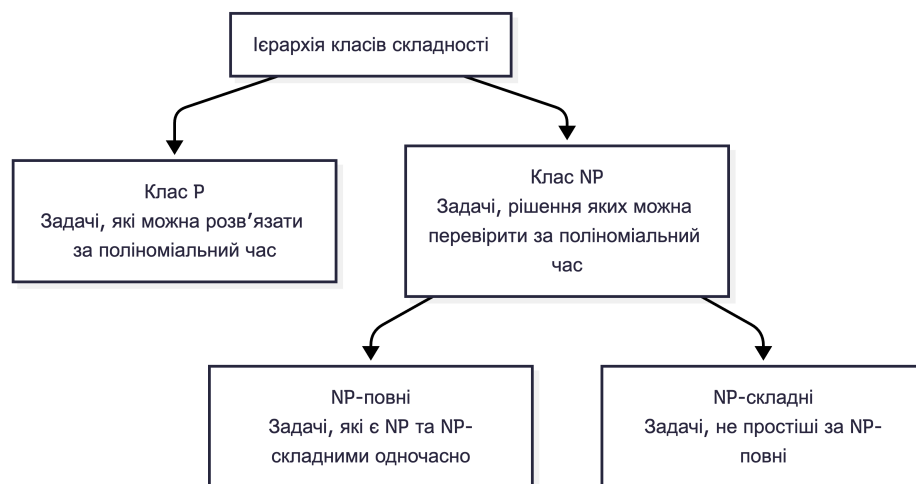


Рис. 1.5. Ієрархія класів складності задач

Задачі класу NP поділяють на NP -складні та NP -повні. Задача B називається NP -складною, якщо для будь-якої задачі A з класу NP існує поліноміальне зведення A до B , тобто задача B є не менш складною за будь-яку задачу з класу NP . NP -повною називається задача, яка одночасно належить до класу NP і є NP -складною [109], при цьому такі задачі характеризують межу складності в межах класу NP , оскільки у разі існування поліноміального алгоритму для хоча б однієї NP -повної задачі це означатиме, що всі задачі класу NP можуть бути розв'язані за поліноміальний час, тобто $P = NP$.

Розв'язання задач класу NP-повних відноситься до відкритих проблем теорії обчислювальної складності [63].

Прикладом NP-повної задачі є задача про задоволення булевих формул, що було доведено теоремою Кука-Левіна [78], при цьому задача максимального розрізу також належить до класу NP-складних задач [2]. Точні методи розв'язання задач оптимізації забезпечують знаходження глобально оптимального розв'язку для заданої задачі, однак їх застосування для NP-складних задач пов'язане зі значними обчислювальними витратами, що обмежує їх ефективність задачами невеликого розміру або задачами зі специфічною структурою. До основних методів точного розв'язання відносять (рис. 1.6): метод гілок і меж, метод гілок, відсічень і цін, цілочисельне лінійне програмування та динамічне програмування.

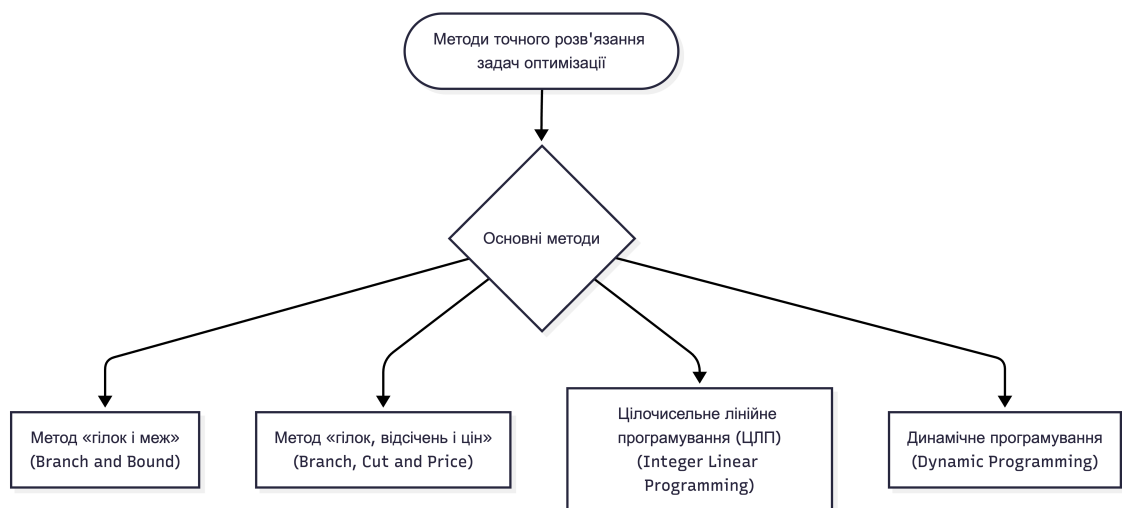


Рис. 1.6. Методи розв'язання задач оптимізації

Метод гілок і меж є загальною схемою розв'язання задач комбінаторної оптимізації та цілочисельного програмування, у межах якої вихідна задача розбивається на підзадачі (гілки), а оцінки меж використовуються для відсікання тих підзадач, які не можуть містити оптимального розв'язку, що дозволяє зменшити розмір простору пошуку [94]. Метод гілок, відсічень і цін є узагальненням методу гілок і меж, яке поєднує генерацію відсікаючих площин для посилення оцінок меж та генерацію стовпців для роботи з великою кількістю змінних [123]. Цілочисельне лінійне програмування (ILP)

охоплює широкий клас задач комбінаторної оптимізації, які формалізуються у вигляді лінійних моделей із цілочисельними змінними, при цьому для їх розв'язання застосовуються спеціалізовані програмні засоби, що базуються на методах гілок і меж та гілок і відсічень [79]. Динамічне програмування застосовується до задач, що допускають декомпозицію на підзадачі з перекривною структурою, при цьому результати розв'язання підзадач зберігаються з метою уникнення повторних обчислень [1]. Зазначені методи використовуються у програмних системах для розв'язання задач комбінаторної оптимізації великої розмірності, зокрема таких, як задача комівояжера або задача максимального розрізу графа.

Оскільки значна частина задач оптимізації належить до класу NP-складних, а їх точне розв'язання за поліноміальний час у загальному випадку не є відомим (за припущення $P \neq NP$), для таких задач застосовуються апроксимаційні алгоритми. Апроксимаційний алгоритм визначається як алгоритм, що працює за поліноміальний час і забезпечує розв'язок, значення якого знаходиться у межах заданого коефіцієнта від оптимального [109]. Якість апроксимаційних алгоритмів оцінюється за коефіцієнтом апроксимації та твердістю апроксимації. Коефіцієнт апроксимації α кількісно характеризує найгірше відхилення розв'язку, отриманого алгоритмом, від оптимального значення; для задач мінімізації виконується умова $\alpha \geq 1$, тоді як для задач максимізації визначення коефіцієнта залежить від обраної конвенції, при цьому отриманий розв'язок відрізняється від оптимального не більше ніж у α разів (для задач мінімізації) або не менше ніж у відповідному відношенні (для задач максимізації) [109]. Твердість апроксимації характеризує обмеження на досяжні значення коефіцієнта апроксимації для окремих NP-складних задач, показуючи, що досягнення певного рівня наближення є обчислювально складним і у ряді випадків еквівалентним за складністю знаходженню точного розв'язку [63]. Це означає, що для таких задач не існує апроксимаційних алгоритмів із гарантованим коефіцієнтом, який забезпечує доволно мале відхилення від оптимального значення за поліноміальний час за умови, що

$P \neq NP$. Зокрема, для задач пошуку кліки максимального розміру (Max-Clique) та визначення хроматичного числа графа (Chromatic Number) отримано результати, що встановлюють жорсткі обмеження на можливість їх ефективної апроксимації [111]. Розробка апроксимаційних алгоритмів розглядається як один із підходів до розв'язання складних задач оптимізації у випадках, коли знаходження точного оптимального розв'язку є обчислювально недоцільним.

Задача максимального розрізу належить до класу задач комбінаторної оптимізації та теорії графів і відноситься до класу NP-складних проблем, що вказує на відсутність відомих алгоритмів її точного розв'язання за поліноміальний час у загальному випадку для графів довільного розміру [86]. Суть задачі полягає у розбитті множини вершин графа на дві підмножини таким чином, щоб кількість або сумарна вага ребер, які з'єднують вершини з різних підмножин, була максимальною [43]. Дати означення задачі можна наступним чином: нехай дано неорієнтований граф $G = (V, E)$, де V – це множина вершин, а E – множина ребер. Кожне ребро $(u, v) \in E$ може мати вагу більшу за 0 [90]. Якщо ваги не задані, передбачається, що всі ребра мають вагу 1. Класичний розв'язок задачі на графі з п'ятьма вершинами можна побачити на рис. 1.7.

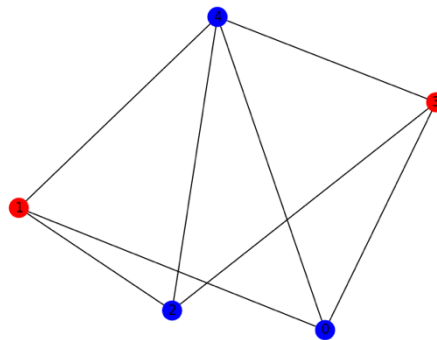


Рис. 1.7. Приклад максимального розрізу на графі з п'ятьма вершинами

Розріз $E(V_1, V_2)$ складається з усіх ребер, що з'єднують вершину з V_1 з вершиною V_2 , завдання полягає в тому, щоб знайти таке розбиття V_1, V_2 , при якому сума ваг ребер, що перетинають розріз, буде максимальною [40]. Задача

Max-Cut може бути сформульована як задача бінарної квадратичної оптимізації [33]. Для кожної вершини $i \in V$ введемо бінарну змінну x_i , яка набуває значення 1 або -1. Якщо $x_i = 1$, вершина i належить до V_1 , інакше $x_i = -1$, вершина i належить до V_2 . Тоді для будь-якого ребра $(i, j) \in E$, якщо вершини i та j знаходяться в різних підмножинах, їхні відповідні x_i та x_j матимуть різні знаки, і тоді $x_i = x_j = -1$. Якщо вони в одній підмножині, $x_i = x_j = 1$. Таким чином, значення $\frac{1-x_i x_j}{2}$ дорівнює 1, якщо ребро (i, j) перетинає розріз, і $= 0$, якщо ні. Мета задачі максимального розрізу максимізувати суму ваг ребер, що перетинають розріз:

$$\text{Maximize } \sum_{(i,j) \in E} w_{ij} \left(\frac{1-x_i x_j}{2} \right), \quad (1.1)$$

де $x_i \in -1, 1$ для всіх $i \in V$. Ця формалізація підкреслює складність задачі, оскільки простір пошуку зростає експоненціально з кількістю вершин.

Задача Max-Cut застосовується у різних прикладних галузях (рис. 1.8), зокрема в аналізі соціальних мереж та кластеризації даних, де вона використовується для виявлення спільнот та групування елементів за певними ознаками [105].

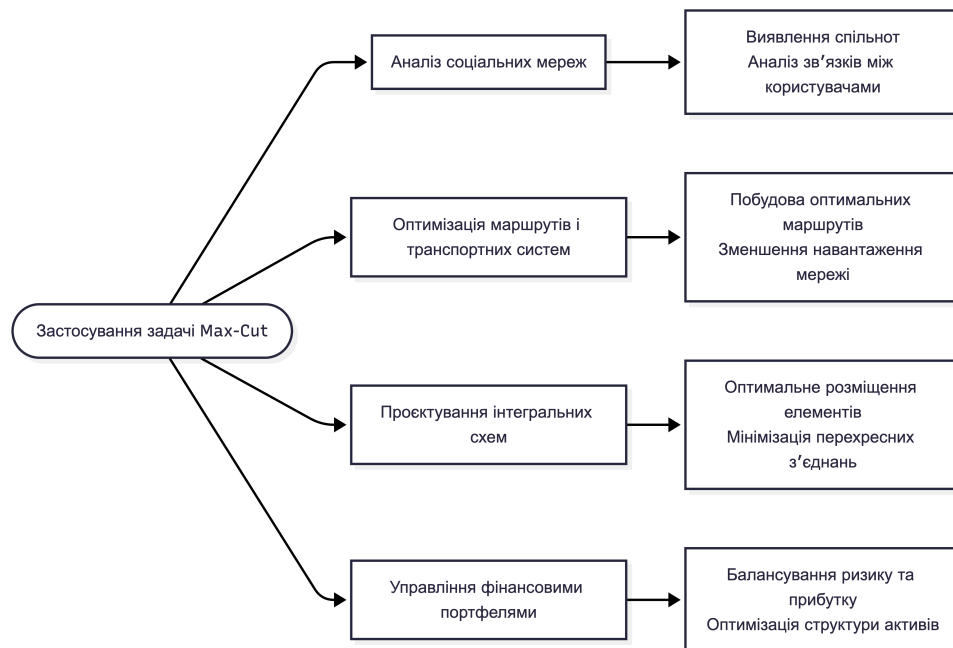


Рис. 1.8. Приклади застосування задачі максимального розрізу

У проєктуванні інтегральних схем задача Max-Cut використовується при розв’язанні задач розміщення компонентів та зменшення кількості перетинів провідників, що враховується при оптимізації відповідних характеристик системи, тоді як у транспортних системах і логістиці вона застосовується для поділу мережі з метою оптимізації маршрутів або інших функціональних параметрів [29]. У фінансовому моделюванні задача Max-Cut розглядається у контексті задач оптимізації портфеля, зокрема при моделюванні співвідношення ризику та прибутковості [39], а у галузі кібербезпеки – при аналізі мережових структур, включаючи задачі сегментації та виявлення потенційних вразливостей [24]. Крім того, задача Max-Cut є еквівалентною задачі знаходження основного стану моделі Ізінга, що використовується у статистичній фізиці для дослідження властивостей магнітних систем. Зазначені прикладні аспекти зумовлюють інтерес до розробки алгоритмічних підходів до розв’язання задачі Max-Cut, зокрема з урахуванням її належності до класу NP-складних задач, що обмежує можливість застосування точних методів для графів великої розмірності [105]. У зв’язку з цим для задачі Max-Cut використовуються як точні методи, що застосовуються до задач невеликого розміру, так і апроксимаційні та евристичні алгоритми, які дозволяють отримувати наближені розв’язки для задач більшої розмірності.

Метод повного перебору є одним із точних підходів до розв’язання задачі максимального розрізу, який передбачає перевірку всіх можливих розбиттів множини вершин графа на дві підмножини (рис. 1.9). Для графа з n вершинами вершинами кількість унікальних розбиттів становить 2^{n-1} , оскільки порядок підмножин не враховується, а одну з вершин можна зафіксувати в одній із підмножин. З урахуванням експоненційного зростання кількості варіантів розбиття, застосування даного підходу обмежується задачами з невеликою кількістю вершин.

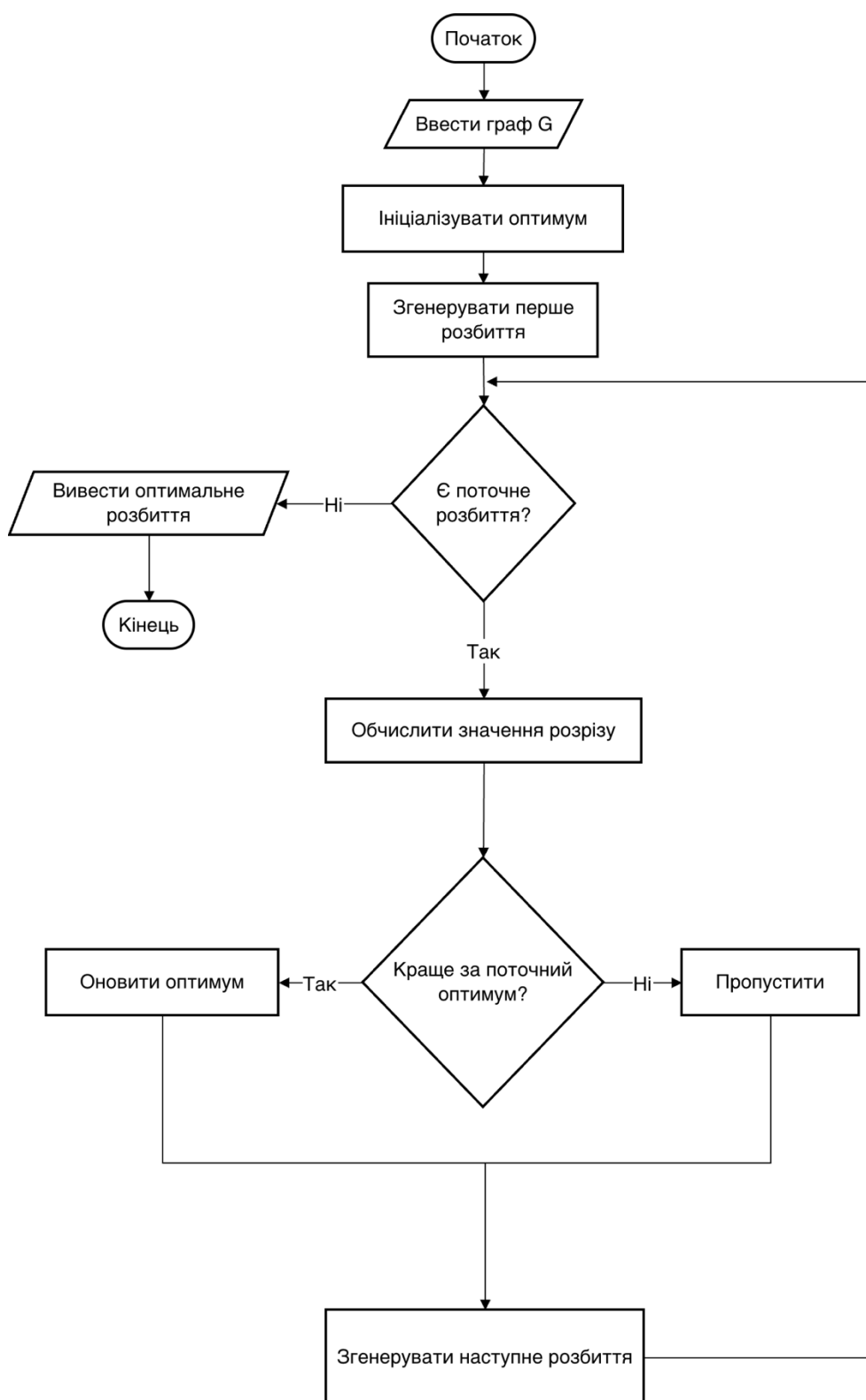


Рис. 1.9. Схема процесу методу повного перебору

Метод гілок і меж є одним із підходів до розв’язання задач комбінаторної оптимізації та цілочисельного програмування, який застосовується для знаходження точного (глобально оптимального) розв’язку, у тому числі для задач, що належать до класу NP-складних, шляхом

систематичного дослідження простору можливих розв’язків. Суть методу полягає (рис. 1.10) у послідовному розбитті вихідної задачі на підзадачі та використанні оцінок меж для відсікання тих частин простору пошуку, які не містять оптимального розв’язку [119].

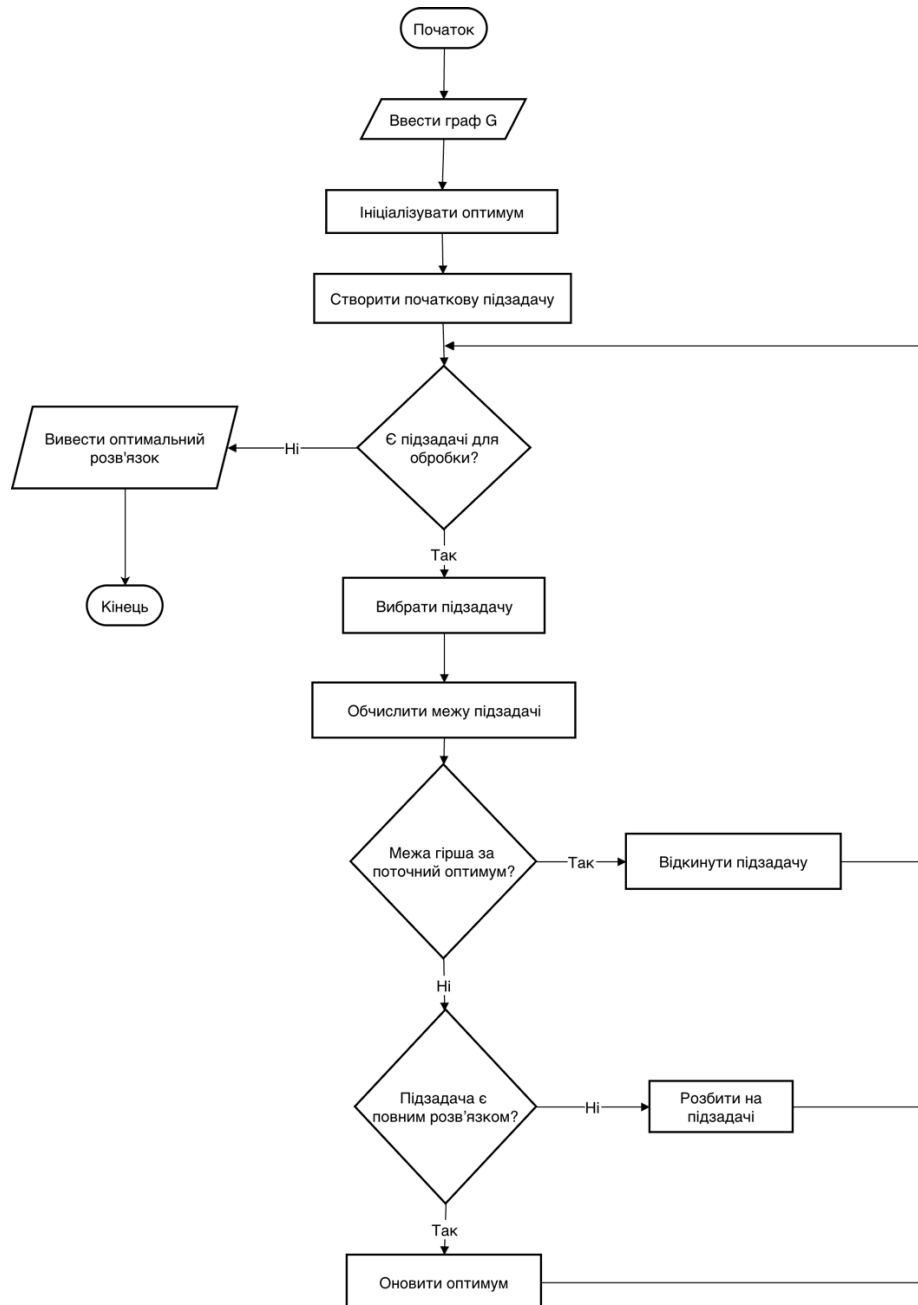


Рис. 1.10. Схема роботи методу «гілок та меж»

Серед ключових компонентів методу гілок і меж виділяють розгалуження, оцінювання та відсікання. На початковому етапі метод рекурсивно розбиває простір можливих розв’язків на менші, непересічні підпростори [119], формуючи дерево пошуку, у якому кожен вузол відповідає

окремій підзадачі, а переходи до дочірніх вузлів відображають подальше розбиття простору розв'язків. У контексті задачі розподілу об'єктів на дві множини процедура розгалуження може бути реалізована шляхом фіксації належності окремих елементів до різних підмножин, що призводить до формування відповідних підзадач. Для кожної підзадачі, яка відповідає вузлу дерева пошуку, обчислюються оцінки меж оптимального значення. Для задач максимізації визначається верхня межа – значення, яке оптимальний розв'язок у межах підзадачі не може перевищити, тоді як для задач мінімізації визначається нижня межа – значення, менше за яке оптимальний розв'язок не може бути отриманий. Отримані оцінки використовуються для порівняння підзадач між собою та з поточним найкращим розв'язком [1]. Етап відсікання полягає у виключенні з подальшого розгляду тих підпросторів (гілок дерева пошуку), для яких на основі обчислених меж встановлено, що вони не містять оптимального розв'язку. Зокрема, для задач мінімізації, якщо нижня межа для певної підзадачі перевищує значення найкращого на даний момент розв'язку, така підзадача виключається з розгляду, оскільки її оптимальне значення не може покращити вже отриманий результат. Застосування процедури відсікання дозволяє зменшити обсяг простору пошуку порівняно з повним перебором усіх можливих розв'язків [50].

Алгоритм Гоєманса-Вільямсона (Goemans–Williamson, GW) є апроксимаційним алгоритмом для задачі Max-Cut, який забезпечує знаходження розв'язку з гарантованим відношенням до оптимального значення [56]. Для даного алгоритму встановлено коефіцієнт апроксимації 0.878, який відповідає найкращим відомим теоретичним оцінкам для цієї задачі. При цьому застосування алгоритму Гоєманса-Вільямсона не змінює належності задачі Max-Cut до класу NP-складних і не забезпечує отримання точного оптимального розв'язку у загальному випадку. Основні етапи алгоритму GW (рис. 1.11) включають формулювання задачі Max-Cut у вигляді задачі квадратичної оптимізації з подальшим перетворенням у задачу напіввизначеного програмування (SDP).



Рис. 1.11. Схема основних кроків роботи алгоритму GW

Ця релаксація передбачає заміну бінарних змінних $x_i \in \{-1, 1\}$ на вектори $v_i \in R^n$, розташовані на одиничній сфері, таким чином, що їх скалярний добуток $v_i \cdot v_j$ апроксимує $x_i x_j$. Метою цього етапу є максимізація $\frac{1}{2} \sum_{i,j \in E} w_{ij}$ [37]. Далі, отримане SDP розв'язується за допомогою стандартних чисельних методів, що дозволяє отримати набір векторів v_i , оскільки розв'язок SDP представлений векторами, а не бінарними значеннями, виконується процедура випадкового округлення, вона полягає в генерації випадкової гіперплощини, що проходить через початок координат [11]. Вершини, вектори яких розташовані по один бік від цієї гіперплощини, відносяться до однієї підмножини розрізу, а ті, що по інший бік – до іншої, це досягається шляхом вибору випадкового вектора $r \in R^n$ з рівномірним розподілом на одиничній сфері, і тоді $x_i = \text{sgn}(v_i \cdot r)$ [107]. Таким чином, алгоритм Гоєманса-Вільямсона ілюструє можливість побудови поліноміальних апроксимаційних алгоритмів для NP-складних задач, які забезпечують гарантоване відношення отриманого розв'язку до оптимального значення, проте обчислювальні

витрати, пов'язані з розв'язанням задач напіввизначеного програмування для графів великої розмірності, можуть бути суттєвими [107].

Жадібні алгоритми формують розв'язок шляхом послідовного вибору локально оптимальних рішень на кожному кроці, при цьому передбачається, що така стратегія може привести до наближеного глобального розв'язку [64]. Для задачі Max-Cut простий жадібний підхід (рис. 1.12) може бути реалізований шляхом початкового віднесення всіх вершин до однієї підмножини з подальшим ітеративним переміщенням окремих вершин до іншої підмножини у випадку, якщо це призводить до збільшення значення розрізу [118]. Зазначені алгоритми характеризуються невисокою обчислювальною складністю, однак їх застосування пов'язане з ризиком отримання локально оптимальних розв'язків, що може призводити до відхилення від глобального оптимуму та відсутності гарантованої якості результату для задачі Max-Cut [64].

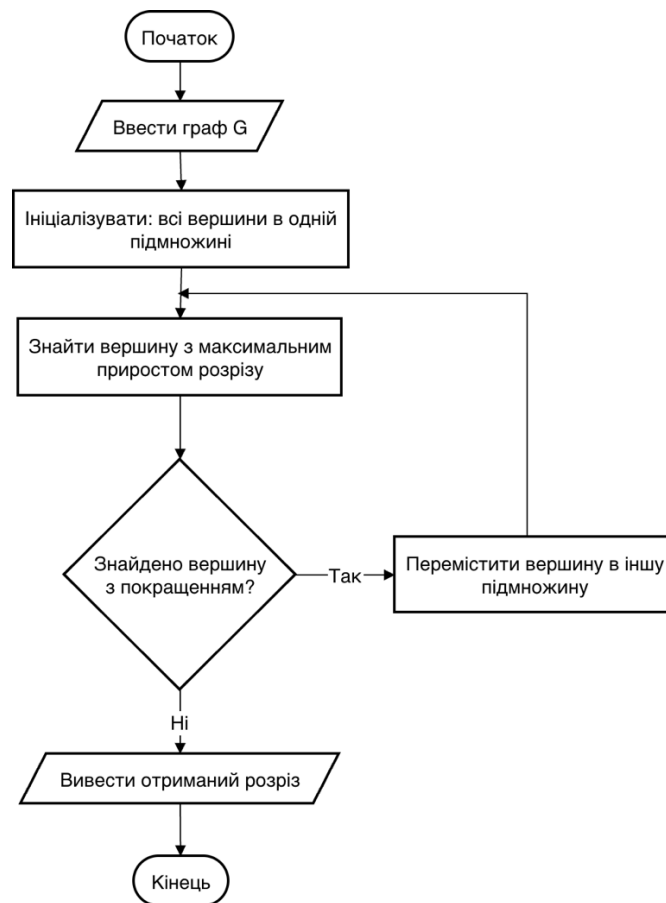


Рис. 1.12. Схема простого жадібного алгоритму для Max-Cut

Методи локального пошуку базуються на використанні початкового розв'язку, який послідовно модифікується шляхом переходу до сусідніх розв'язків, що визначаються незначними змінами поточного стану [72]. Метод локального пошуку (рис. 1.13) починається з побудови початкового, зокрема випадкового, розрізу [9], після чого на кожній ітерації здійснюється аналіз множини сусідніх розв'язків. У випадку виявлення сусіднього розв'язку, який покращує поточне значення функції Max-Cut, виконується перехід до відповідного стану, причому ітераційний процес продовжується доти, доки існують сусідні розв'язки, що забезпечують покращення, а його завершення відбувається за відсутності таких переходів. Методи локального пошуку характеризуються можливістю досягнення локально оптимальних розв'язків, однак їх застосування пов'язане з обмеженням, що полягає у потенційному потраплянні в локальні оптимуми, подібно до жадібних алгоритмів, при цьому у ряді випадків такі підходи дозволяють отримувати кращі результати порівняно з базовими жадібними стратегіями [9].

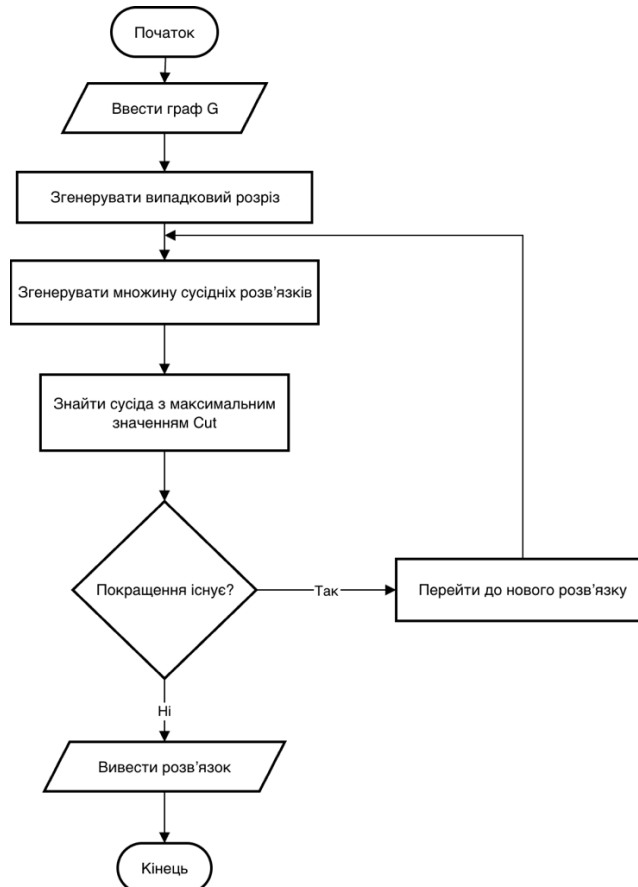


Рис. 1.13. Схема роботи методу локального пошуку для Max-Cut

Генетичні алгоритми належать до класу метаевристичних методів, побудованих на основі принципів біологічної еволюції, та застосовуються для розв’язання складних задач оптимізації, зокрема задачі Max-Cut [41]. Принцип функціонування генетичних алгоритмів (рис. 1.14) передбачає послідовність етапів, що включають формування початкової популяції, оцінювання розв’язків та застосування операторів еволюції. На початковому етапі формується популяція випадкових розв’язків, при цьому у контексті задачі Max-Cut кожна «хромосома» може бути представлена у вигляді бінарного вектора, який відображає розподіл вершин графа між двома підмножинами [97].

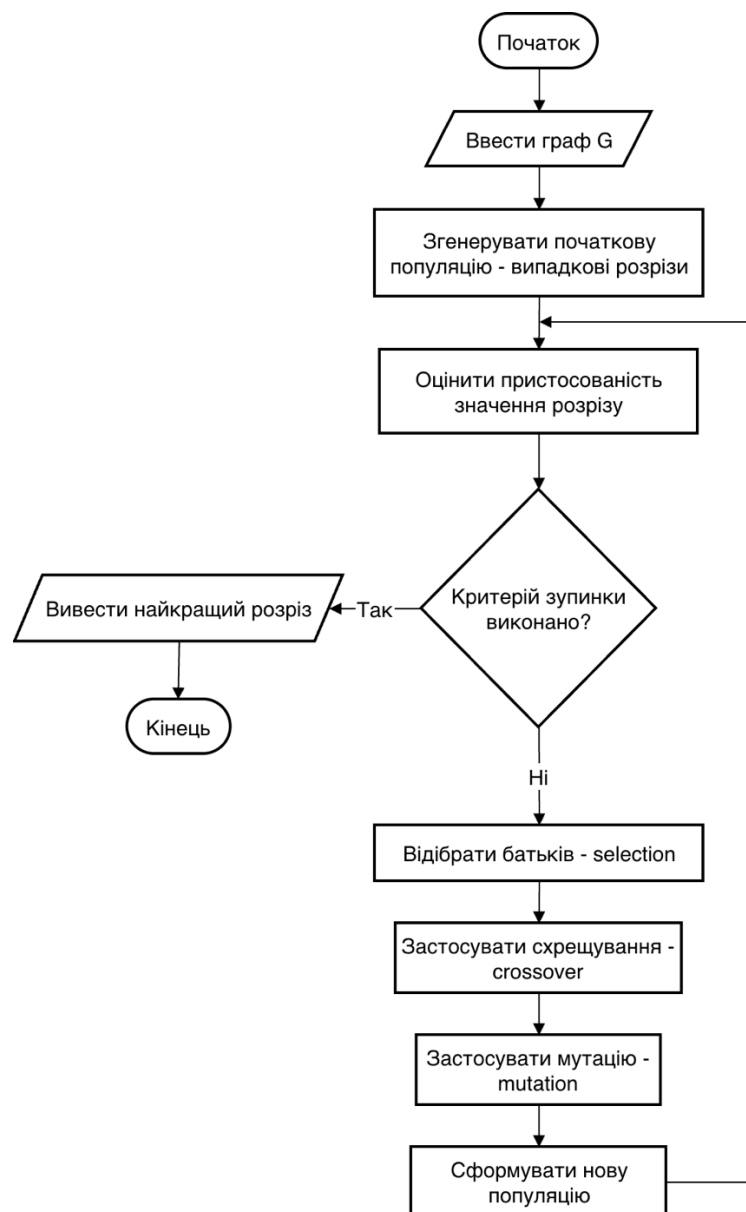


Рис. 1.14. Схема роботи генетичного алгоритму для Max-Cut

Для кожної хромосоми визначається значення функції пристосованості, яке відображає якість відповідного розв'язку, після чого на основі отриманих оцінок здійснюється відбір хромосом для формування наступного покоління, при цьому перевага надається розв'язкам із вищими значеннями функції пристосованості. Відібрані хромосоми піддаються операції схрещування, у межах якої відбувається обмін генетичним матеріалом між батьківськими особинами з метою формування нових розв'язків, тоді як до отриманих нащадків застосовується операція мутації, що передбачає внесення випадкових змін для підтримання різноманіття популяції та зменшення ймовірності передчасної збіжності до локальних оптимумів [112]. Описаний процес повторюється протягом заданої кількості поколінь або до досягнення критерію зупинки, при цьому генетичні алгоритми застосовуються для дослідження простору розв'язків великої розмірності та отримання наближених розв'язків задач оптимізації [124].

Імітація відпалу є метаевристичною технікою оптимізації, яка базується на аналогії з процесом термічної обробки металів, у межах якого матеріал піддається нагріванню з подальшим поступовим охолодженням з метою досягнення стабільного стану з мінімальною кількістю дефектів [115]. Алгоритм імітації відпалу був запропонований Кіркпатріком С. (Kirkpatrick S.), Геллаттом К. (Gelatt C.) та Веккі М. (Vecchi M.) у 1983 році та застосовується для розв'язання задач комбінаторної оптимізації, зокрема задачі максимального розрізу.

Основний принцип роботи алгоритму імітації відпалу (рис. 1.15) полягає у дослідженні простору розв'язків із можливістю переходу з локальних оптимумів за рахунок прийняття гірших розв'язків із певною ймовірністю [21]. Алгоритм починається з ініціалізації випадкового початкового розв'язку та встановлення високого значення температури, після чого на кожній ітерації генерується сусідній розв'язок, який приймається у випадку покращення значення цільової функції [84]. Якщо згенерований сусідній розв'язок є гіршим, його прийняття можливе з ймовірністю, що

визначається відповідно до розподілу Больцмана, причому при вищих значеннях температури збільшується ймовірність прийняття таких розв'язків, що сприяє більш повному дослідженню простору розв'язків [99]. Після цього температура поступово зменшується відповідно до заданого графіка охолодження, при цьому процес охолодження визначає співвідношення між дослідженням простору розв'язків та переходом до стабільних станів.

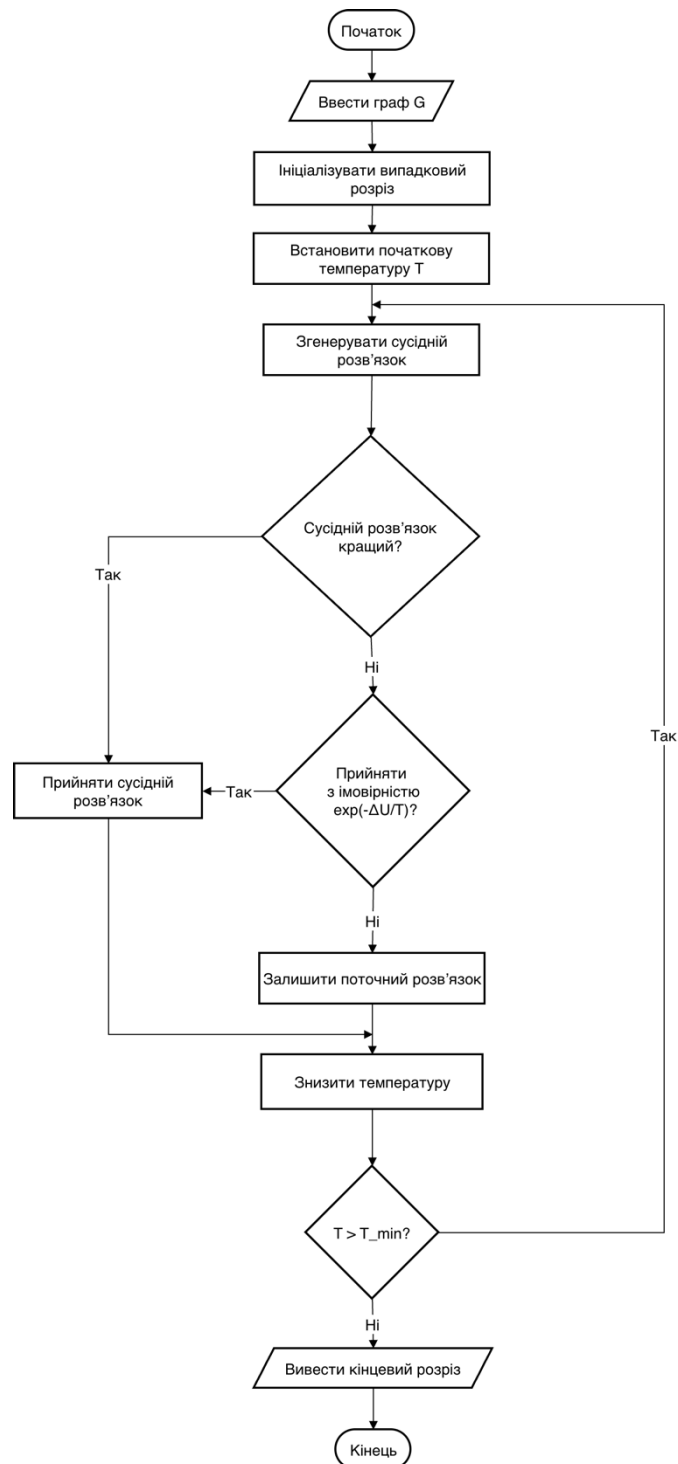


Рис. 1.15. Схема принципу роботи імітації відпалу для Max-Cut

Процес завершується, коли температура досягає заданого мінімального значення або після виконання визначеної кількості ітерацій. Застосування алгоритму імітації відпалу до задач комбінаторної оптимізації пов'язане з необхідністю налаштування параметрів, зокрема вибору графіка охолодження, при цьому обчислювальні витрати можуть зростати залежно від параметрів алгоритму та розмірності задачі [99].

На основі проаналізованої інформації було створено зведену таблицю 1.1.

Таблиця 1.1

Класичні алгоритми для розв'язання задачі максимального розрізу

Метод	Тип	Переваги	Недоліки
Перебір	Точний	Забезпечує знаходження глобально оптимального розв'язку	Експоненційна складність
Метод «гілок і меж»	Точний	Забезпечує знаходження оптимального розв'язку за рахунок скорочення простору пошуку порівняно з повним перебором	Значні обчислювальні витрати для задач великої розмірності
Алгоритм Гоєманса-Вільямсона	Апроксимаційний	Забезпечує гарантований коефіцієнт апроксимації 0.878 для невід'ємних ваг ребер	Обчислювальні витрати, пов'язані з розв'язанням задач напіввизначеного програмування, зростають зі збільшенням розміру графа
Жадібні алгоритми	Евристичний	Простота реалізації та невисока обчислювальна складність	Можливість отримання локально оптимальних розв'язків
Локальний пошук	Евристичний	Застосовується для отримання наближених розв'язків за обмежений час	Потрапляння у локальні оптимуми

Таблиця 1.1 продовження

Метод	Тип	Переваги	Недоліки
Генетичні алгоритми	Евристичний	Забезпечують дослідження простору розв'язків великої розмірності	Значні обчислювальні витрати та необхідність налаштування параметрів
Імітація відпалу	Евристичний	Дозволяє уникати локальних оптимумів за рахунок імовірного прийняття гірших розв'язків	Залежність від параметрів алгоритму, зокрема графіка охолодження

Для розв'язання задачі максимального розрізу графа на класичних обчислювальних системах використовується широкий спектр алгоритмічних підходів, які відрізняються за точністю отриманих розв'язків та обчислювальною складністю. Незважаючи на різноманітність існуючих методів, належність задачі Max-Cut до класу NP-складних обмежує можливість її точного розв'язання для графів великої розмірності за поліноміальний час. У зв'язку з цим значна увага приділяється розробці альтернативних підходів, зокрема квантово-класичних алгоритмів, які поєднують можливості квантових обчислень із класичними методами обробки даних.

1.3. Квантові та гібридні підходи до вирішення Max-Cut

Квантові комп'ютери використовують принципи квантової механіки для виконання обчислень, що створює передумови для отримання обчислювальної переваги над класичними обчислювальними системами, при цьому, на відміну від класичних комп'ютерів, які оперують окремими бінарними станами, квантові системи дозволяють застосовувати оператори до суперпозиції бінарних рядків, що забезпечує можливість одночасного опрацювання множини станів [66]. Зазначена властивість використовується у квантових алгоритмах для досягнення прискорення обчислень у порівнянні з класичними

підходами для окремих класів задач [66]. Ключовими концепціями квантових обчислень є кубіти, суперпозиція, квантова заплутаність та квантовий паралелізм (рис. 1.16).

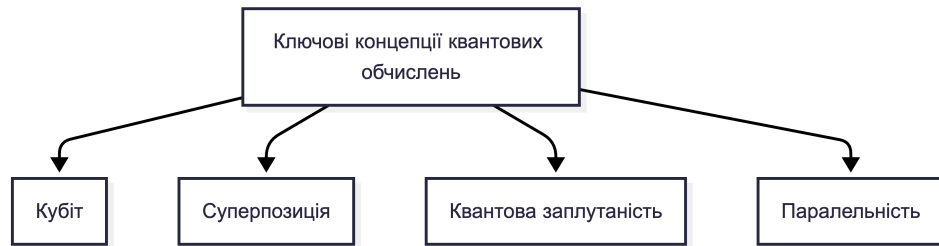


Рис. 1.16. Ключові концепції квантових обчислень

Квантові біти (кубіти) є базовими одиницями квантової інформації, при цьому, на відміну від класичних бітів, які можуть перебувати лише у станах 0 або 1, кубіт може знаходитися у суперпозиції цих станів, тобто описується як лінійна комбінація базисних станів $|0\rangle$ та $|1\rangle$ [25]. Графічне представлення стану кубіта здійснюється за допомогою сфери Блоха, тоді як класичний біт відповідає окремим точкам, що відображає обмеженість його можливих станів (рис. 1.17).

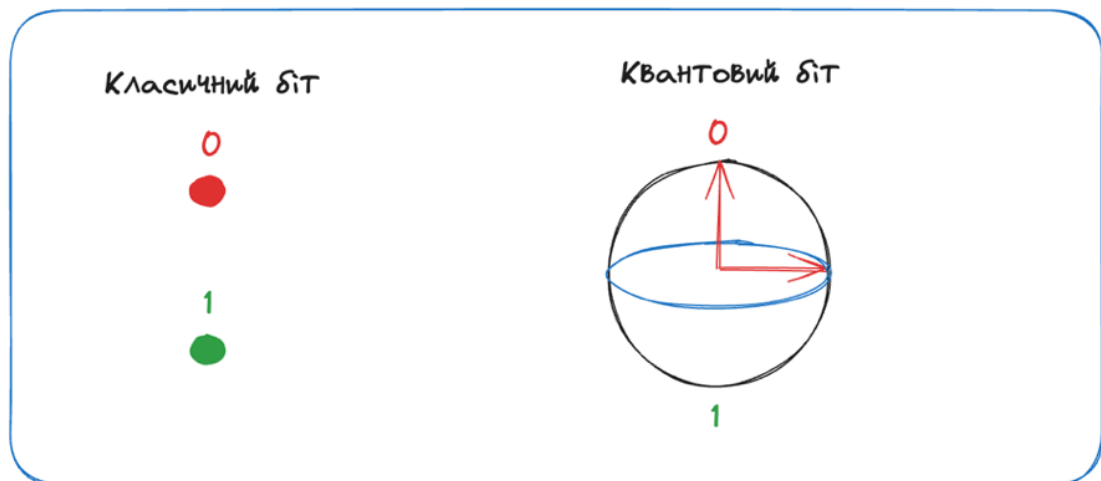


Рис. 1.17. Графічне зображення біта та кубіта [116]

Математично кубіт описується як одиничний вектор у двовимірному гільбертовому просторі $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$, де α та β є комплексними числами, що задовольняють умову нормалізації $|\alpha|^2 + |\beta|^2 = 1$ [25]. При переході до систем із кількох кубітів розмірність простору станів зростає експоненційно

відносно кількості кубітів, що зумовлює можливість одночасного представлення великої кількості конфігурацій станів [8].

Суперпозиція – є фундаментальною властивістю квантових систем, яка полягає у можливості перебування кубіта у лінійній комбінації базисних станів, унаслідок чого система з N кубітів описується у просторі розмірності 2^N , що використовується у квантових алгоритмах для обробки множини станів [122]. При вимірюванні квантового стану відбувається його редукція (колапс) до одного з базисних станів із ймовірностями, що визначаються квадратами модулів коефіцієнтів α та β [108].

Для ілюстрації властивостей суперпозиції та процесу вимірювання у квантовій механіці використовується уявний експеримент Шредінгера Е. (Schrödinger E.), запропонований у 1935 році [23], у якому розглядається макроскопічна система, стан якої пов'язаний із квантовою подією, що дозволяє продемонструвати принципову відмінність між квантовим описом системи до моменту вимірювання та результатом спостереження. У межах цього експерименту стан системи до моменту спостереження описується як суперпозиція альтернативних результатів, тоді як акт вимірювання приводить до фіксації одного з можливих станів [100]. Таким чином, суперпозиція та пов'язаний із нею процес вимірювання визначають специфіку обробки інформації у квантових обчислювальних системах, відрізняючи їх від класичних підходів до обчислень [136].

Квантова запутаність є ще одним фундаментальним явищем квантової механіки, яке описує стан системи з кількох кубітів, у якому їхні квантові стани не можуть бути представлені незалежно один від одного (рис. 1.18) [76]. У такому випадку стан всієї системи визначається як єдине ціле, а результати вимірювання окремих кубітів виявляються статистично корельованими незалежно від просторового розділення між ними [92].

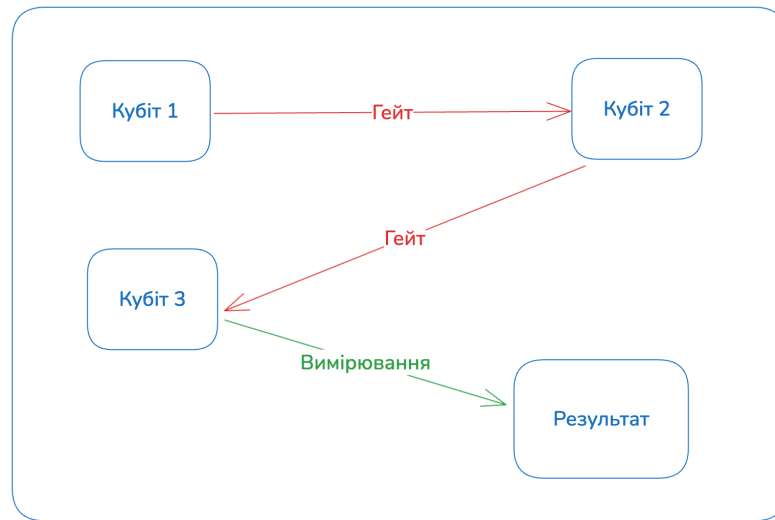


Рис. 1.18. Квантове заплутування

Зазначена кореляція відсутня у класичних системах і розглядається як один із ресурсів квантових обчислень, що використовується у квантових алгоритмах для досягнення прискорення обчислень у порівнянні з класичними підходами, зокрема в алгоритмі Шора для факторизації цілих чисел та алгоритм Гровера для пошуку в неструктурованих базах даних [37].

Квантові вентиля (квантові гейти) є базовими елементами квантових схем, які забезпечують виконання операцій над кубітами та реалізацію квантових алгоритмів, при цьому вони розглядаються як аналог логічних вентилів у класичних обчисленнях, але діють у просторі квантових станів із використанням принципів суперпозиції та заплутаності [122]. Квантові гейти класифікують за кількістю кубітів на *однокубітні* та *багатокубітні* вентиля.

На рисунку 1.19 наведено класифікацію однокубітних квантових вентилів за типами виконуваних перетворень. Базові вентиля (X, Y, Z, H) реалізують елементарні операції над квантовими станами, фазові вентиля (S, T) забезпечують зміну відносної фази амплітуд, тоді як вентиля обертання $R_X(\theta)$, $R_Y(\theta)$, $R_Z(\theta)$ та $U(\theta, \varphi, \lambda)$ здійснюють параметризовані перетворення стану кубіта на сфері Блоха [13].

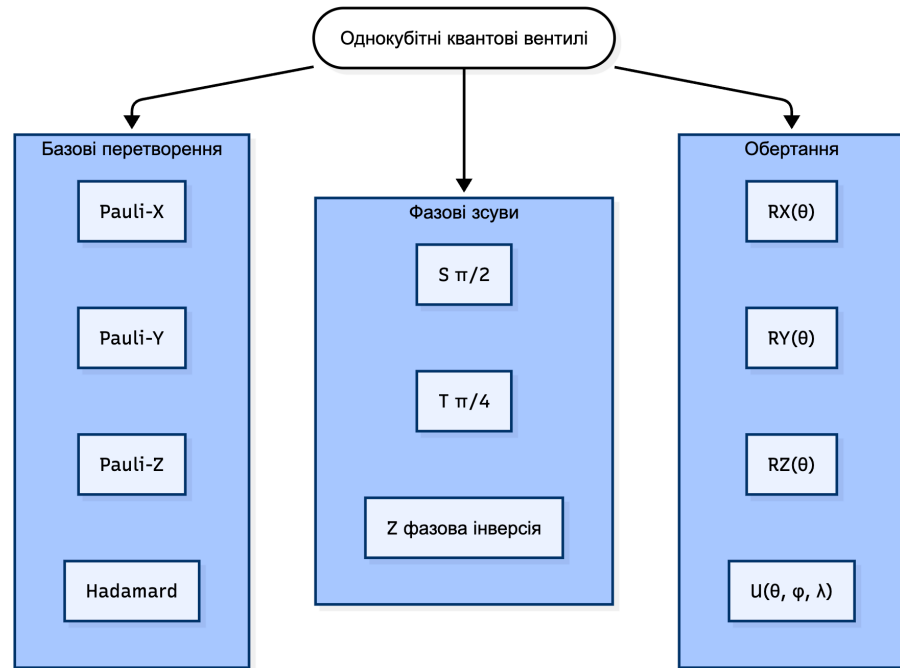


Рис. 1.19. Однокубітні квантові вентиля

Найбільш елементарним однокубітним гейтом для маніпуляції станами кубіта є гейт *Паулі-Х* (X), матриця якого визначається як $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, цей квантовий вентиль здійснює інверсію стану кубіта, перетворюючи $|0\rangle$ на $|1\rangle$ та навпаки [59]. Гейт Паулі-Х відіграє ключову роль як базовий компонент для реалізації логічних операцій і трансформацій станів у квантових схемах, його дія відповідає обертанню на радіан навколо осі X на сфері Блоха (рис. 1.20).

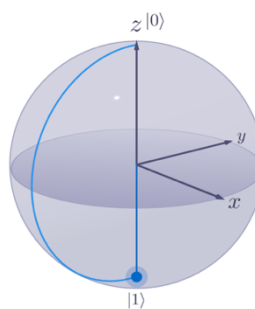


Рис. 1.20. Дія гейта Pauli-X на сфері Блоха [110]

Поворот квантового біта навколо осі Y здійснює гейт *Паулі-У* (Y), поворот можна описати матрицею $\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$, гейт виконує обертання на радіан навколо осі Y , являючи собою комбінацію біт-фліпу та фазового зсуву [102]. Він застосовується для створення суперпозицій із фазовими відмінностями,

входить до повного набору операторів Паулі, вводить уявну фазу та впливає на Y-компоненту спіну (рис. 1.21).

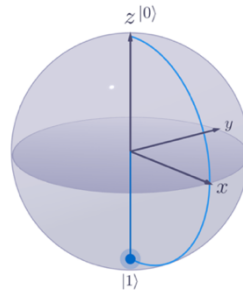


Рис. 1.21. Дія гейта Pauli-Y на сфері Блоха [110]

Третім ключовим оператором Паулі є гейт *Паулі-Z* (Z), що реалізує фазовий зсув кубіта, його матричне представлення задається як $\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$. Вентиль Z вводить фазу до стану π , залишаючи стан незмінним. Він відіграє фундаментальну роль у впровадженні фазових зсувів та умовних операцій у квантових алгоритмах [122]. Дія гейта Паулі-Z відповідає обертанню на радіан навколо осі Z на сфері Блоха, що змінює знак амплітуди стану. Оскільки поворот навколо осі Z не помітний, для наглядності на рис. 1.22 застосовано комбінацію Rx повороту та гейта Z .

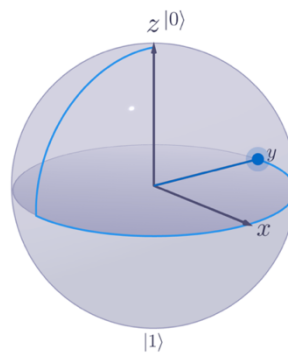


Рис. 1.22. Дія гейта Pauli-Z на сфері Блоха [110]

Для генерації квантової суперпозиції та реалізації принципу квантового паралелізму фундаментальним елементом є гейт *Адамара* (H), матриця якого виглядає як $\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$. Цей вентиль відіграє ключову роль у формуванні суперпозиції станів, перетворюючи $|0\rangle$ на $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$, а $|1\rangle$, у свою чергу, на

$\frac{|0\rangle - |1\rangle}{\sqrt{2}}$ [102]. Він слугує основою для ініціалізації суперпозиції та забезпечення квантової паралельності [4]. Дію гейта на сфері Блоха можна побачити на рис. 1.23.

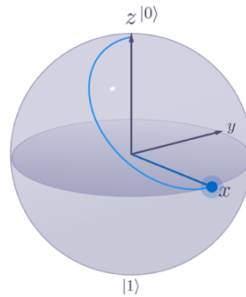


Рис. 1.23. Дія гейта Адамара на сфері Блоха [110]

У квантових обчисленнях для здійснення менших фазових зсувів застосовується фазовий вентиль S , матриця якого має вигляд $\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$. Цей вентиль додає фазу $\pi/2$ до стану $|1\rangle$, що еквівалентно обертанню на 90° навколо осі Z на сфері Блоха (рис. 1.24), він використовується для фазових корекцій у квантових алгоритмах та входить до універсальних наборів вентилів [88].

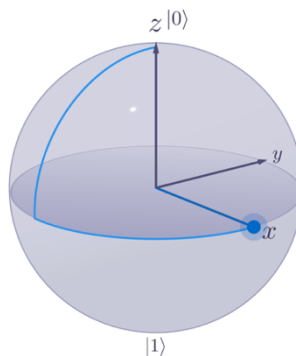


Рис. 1.24. Дія гейта S на сфері Блоха [110]

Для зворотної фазової корекції застосовується спряжений фазовий вентиль $S^\dagger$, матриця якого має вигляд $\begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix}$, причому цей вентиль є ермітовим спряженим до вентиля S і додає фазу $-\pi/2$ до стану $|1\rangle$, а отже еквівалентно виконує обертання на -90° навколо осі Z (рис. 1.25), що

використовується для реалізації зворотних фазових операцій та балансування фаз у квантових схемах [20].

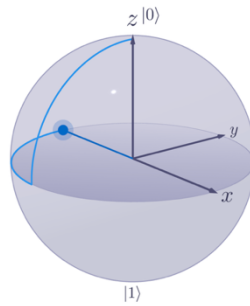


Рис. 1.25. Дія гейта $S^\dagger$ на сфері Блоха [110]

Для забезпечення універсальності квантових обчислень поряд із S-вентилем застосовується T-вентиль, що вводить ще менший фазовий зсув, його матричне представлення має вигляд $\begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$, вентиль додає фазу $\pi/4$ до стану $|1\rangle$, що еквівалентно обертанню на 45° навколо осі Z [20]. Він є ключовим елементом для досягнення універсальності квантових обчислень у наборі, забезпечуючи мінімальний фазовий зсув (рис. 1.26), необхідний для апроксимації будь-яких унітарних операцій.

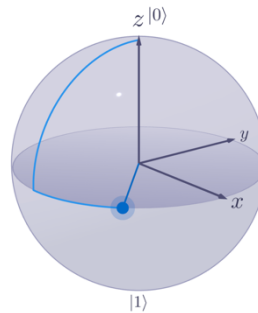


Рис. 1.26. Дія гейта T на сфері Блоха [110]

Подібно до S-вентиля, T-вентиль також має спряжений аналог $T^\dagger$, який виконує зворотну фазову операцію і матриця якого задається як $\begin{pmatrix} 1 & 0 \\ 0 & e^{-i\pi/4} \end{pmatrix}$, цей вентиль додає фазу $-\pi/4$ до стану $|1\rangle$, що еквівалентно обертанню на -45° навколо осі Z (рис. 1.27), гейт використовується для реалізації зворотних операцій в універсальних апроксимаціях квантових схем [91].

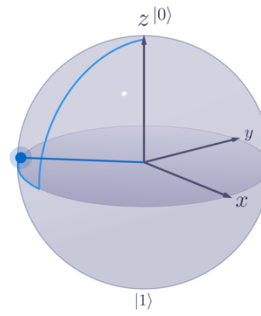


Рис. 1.27. Дія гейта $T^\dagger$ на сфері Блоха [110]

Квантові гейти, що забезпечують реалізацію довільних обертань на сфері Блоха називають *ротаційними*, до них відносять: R_x , R_y та R_z . Вентиль R_x виконує обертання навколо осі X на кут θ , що задається матрицею $\begin{pmatrix} \cos(\theta/2) & -i \sin(\theta/2) \\ -i \sin(\theta/2) & \cos(\theta/2) \end{pmatrix}$, що дозволяє точно обертати стан кубіта навколо осі X на заданий кут θ , сприяючи калібруванню кубітних станів та апроксимації довільних ротацій [132]. Для обертання навколо осі Y застосовується вентиль $R_y(\theta)$ з матрицею $\begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix}$ та реалізує ротацію стану кубіта навколо осі Y на заданий кут θ , він використовується для створення суперпозицій у напрямку осі Y на сфері Блоха [132]. Аналогічно ротаційним гейтам R_x та R_y , гейт $R_z(\theta)$ здійснює обертання стану кубіта навколо осі Z на кут θ , що виражається його матричним представленням $\begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}$, він є фундаментальним для реалізації фазових ротацій у різноманітних квантових алгоритмах, будучи діагональним оператором, він впливає виключно на відносну фазу кубіта [132].

Вентиль $U1(\lambda)$ являє собою спеціалізований фазовий гейт, еквівалентний гейту R_z , він реалізує фазовий зсув, ідентичний до дії гейта R_z та застосовується для простого й ефективного керування фазою [68]. Квантовий гейт $U2(\phi, \lambda)$ реалізує складнішу, але компактну трансформацію, його матричне представлення еквівалентне композиції $R_z(\phi)R_y(\pi/2)R_z(\lambda)$, він застосовується для конструювання компактних квантових схем, які реалізують базові перетворення, та слугує комбінованим ротаційним

блоком [68]. Найуніверсальнішим серед однокубітних вентилів є гейт $U3(\theta, \phi, \lambda)$, здатний реалізовувати будь-яке однокубітне унітарне перетворення, матричне представлення якого еквівалентне композиції $Rz(\phi)Ry(\theta)Rz(\lambda)$ [68]. $U3$ є загальним однокубітним унітарним гейтом, що дозволяє моделювати будь-яку однокубітну операцію, ця універсальність забезпечує застосування для симуляції довільних однокубітних трансформацій.

Найпростішим квантовим гейтом є вентиль ідентичності I , який не змінює стан кубіта, відповідає найпростіша унітарна матриця $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. Цей вентиль зберігає стан кубіта незмінним, виконуючи роль заповнювача в конструкціях складних квантових схем та слугуючи для цілей тестування, як нейтральна операція, він наочно демонструє оборотність квантових перетворень [122].

Багатокубітні вентиля діють на два або більше кубітів(рис. 1.28), дозволяючи реалізувати заплутаність, контрольовані операції та складні алгоритми [97]. Їхня здатність створювати заплутані стани є критично важливою для потужності квантових обчислень [36].

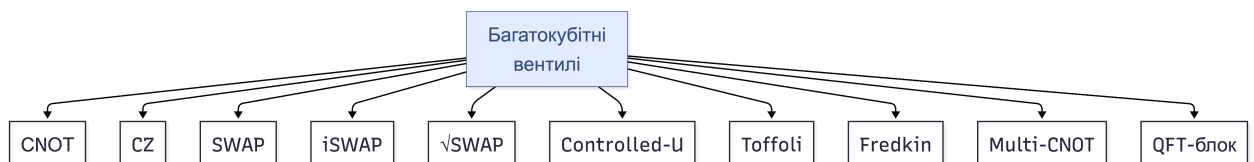


Рис. 1.28. Багатокубітні квантові вентиля

Одним із ключових двокубітних вентилів є $CNOT$, який застосовує логічну NOT-операцію до цільового кубіта лише за умови, що контрольний кубіт перебуває у стані 1, інакше цільовий кубіт залишається незмінним, трансформуючи стан (x, y) на $(x, x \text{ XOR } y)$ [66]. Цей вентиль використовується для реалізації умовної логіки, генерування квантової заплутаності та становить невід’ємну частину універсальних наборів вентилів. Інший двокубітний вентиль CZ , модифікує фазу квантової системи, застосовуючи фазовий зсув $-\pi$ виключно до стану $|11\rangle$, при цьому інші стани

залишаються незмінними, його основне призначення полягає у створенні заплутаності за допомогою фазового зсуву, який залежить від стану контрольного кубіта [101]. Вентиль *SWAP* здійснює обмін станами між двома кубітами, переставляючи їх позиції та перетворюючи стан (x, y) на (y, x) , що робить операцію фундаментальною для встановлення зв'язку між не сусідніми фізичними кубітами та знаходить широке застосування в алгоритмах, таких як квантове перетворення Фур'є та алгоритм Гровера [28]. Вентиль *iSWAP*, також двокубітний, перетворює стани $|01\rangle$ на $-i|10\rangle$ та $|10\rangle$ на $-i|01\rangle$, при цьому стани $|00\rangle$ та $|11\rangle$ залишаються інваріантними, гейт забезпечує обмін станами з додаванням фазового множника $-i$ та часто використовується як потужний заплутуючий вентиль [44]. Вентиль $\sqrt{\text{SWAP}}$, у свою чергу, виконує операцію, що є квадратним коренем від вентиля *SWAP*, і володіє здатністю генерувати заплутаність між двома початково розділеними кубітами, у поєднанні з однокубітними операціями гейт формує універсальний набір вентилів, здатний трансформувати відповідні продуктові стани у максимально заплутані конфігурації [44].

Для реалізації узагальнених умовних операцій використовується вентиль *Controlled-U*, який застосовує унітарну операцію U до цільового кубіта, якщо контрольний кубіт знаходиться у стані $|1\rangle$, в іншому випадку до цільового кубіта застосовується тотожна операція [121]. Серед трикубітних вентилів особливе місце посідає *Toffoli*, який застосовує X-операцію до цільового кубіта тільки тоді, коли обидва контрольні кубіти перебувають у стані 1, таким чином перетворюючи стан (A_1, A_2, A) на $(A_1, A_2, A \text{ XOR } (A_1 \text{ AND } A_2))$, гейт являється універсальним для класичних оборотних обчислень і використовується в таких значних алгоритмах, як алгоритми Шора та Гровера [48]. Вентиль *Fredkin*, також трикубітний, здійснює операцію *SWAP* між двома цільовими кубітами, якщо контрольний кубіт активований, вентиль виконує функцію *controlled-SWAP*, що робить його універсальним у комбінації з однокубітними вентилями та важливим для реалізації квантової корекції

помилки і різноманітних квантових алгоритмів [81]. Розширенням функціональності вентиля *CNOT* є гейт *Multi-CNOT*, що застосовує *X*-операцію до цільового кубіта лише тоді, коли всі n контрольних кубітів одночасно перебувають у стані 1, таким чином представляючи узагальнення вентиля Тоффолі [48]. Важливим оператором, що функціонує з n кубітами, є QFT-блок, який перетворює вхідний квантовий стан з амплітудами, що репрезентують сигнали, у новий стан, де амплітуди кодують дискретне перетворення Фур'є цих сигналів, цей блок є критично важливою підпрограмою в таких алгоритмах, як алгоритм Шора, квантова оцінка фази та квантове моделювання [55].

Багатокубітні вентиля застосовуються у квантових обчисленнях для реалізації взаємодії між кубітами, зокрема для формування станів заплутаності, що використовується у квантових алгоритмах при обробці квантової інформації. Для виконання довільного квантового алгоритму на квантовому процесорі квантова схема приводиться до базового набору вентилів, тобто до множини елементарних операцій, визначеної архітектурою та топологією конкретного квантового пристрою [2]. До таких вентилів відносять, зокрема, вентиль Адамара, Toffoli з двома контрольними кубітами, вентиль Fredkin, а також фазові вентиля T і S , однак у більшості сучасних квантових процесорів реалізація складніших вентилів здійснюється шляхом їх декомпозиції на базові однокубітні та двокубітні операції, що підтримуються апаратною платформою [2].

Використання квантової заплутаності та інтерференції розглядається як один із підходів до побудови алгоритмів, що дозволяють отримувати прискорення обчислень для окремих класів задач [34]. Оскільки задачі комбінаторної оптимізації, як правило, належать до класу NP-складних і характеризуються експоненційним зростанням простору пошуку, вони розглядаються як область застосування квантових обчислювальних методів [47]. У цьому контексті досліджуються варіаційні квантові алгоритми, зокрема квантовий алгоритм наближеної оптимізації (QAOA), які поєднують

класичні методи оптимізації з квантовими обчисленнями для отримання наближених розв'язків задач комбінаторної оптимізації [56].

Квантовий наближений оптимізаційний алгоритм (QAOA) належить до класу гібридних квантово-класичних алгоритмів, що застосовуються для розв'язання задач комбінаторної оптимізації, зокрема в умовах використання шумних квантових пристроїв проміжного масштабу (рис. 1.29) [56]. Основою алгоритму є побудова параметризованої квантової схеми, яка реалізує послідовне застосування двох типів унітарних операторів, що чергуються, а саме оператора вартості (фазового оператора) та мікшерного оператора. Кожен із зазначених операторів залежить від набору параметрів, значення яких визначаються у процесі класичної оптимізації, при цьому задача полягає у знаходженні таких параметрів, за яких вимірювання квантового стану забезпечує отримання розв'язків із максимальним значенням цільової функції або близьких до нього [131].

Для застосування квантового наближеного оптимізаційного алгоритму до задач комбінаторної оптимізації необхідно виконати її формалізацію у вигляді, придатному для квантових обчислень, що передбачає кодування задачі як задачі пошуку бінарного рядка $z = (z_1, z_2, \dots, z_n)$, де $z_i \in 0,1$ або $-1,1$, який мінімізує функцію вартості C [83]. У випадку задачі Max-Cut функція вартості визначається сумою ваг ребер, що перетинають розріз, заданий конфігурацією змінних z . Після кодування, функція вартості C перетворюється на гамільтоніан H_C , який є діагональним в обчислювальному базисі, це означає, що власні значення гамільтоніана безпосередньо відповідають значенням функції вартості для кожного базисного стану [131]. Формування гамільтоніана здійснюється шляхом відображення класичних бінарних змінних z_i на оператори Паулі-Z, таким чином, гамільтоніан можна записати у вигляді $H_C |z\rangle = C|z\rangle$ [126]. QAOA використовує параметризований квантовий анзац, який є послідовністю квантових вентилів, що залежать від наборів параметрів γ та β , анзац складається з p послідовних

шарів (глибини схеми), кожен з яких включає застосування оператора вартості та мікшерного оператора.

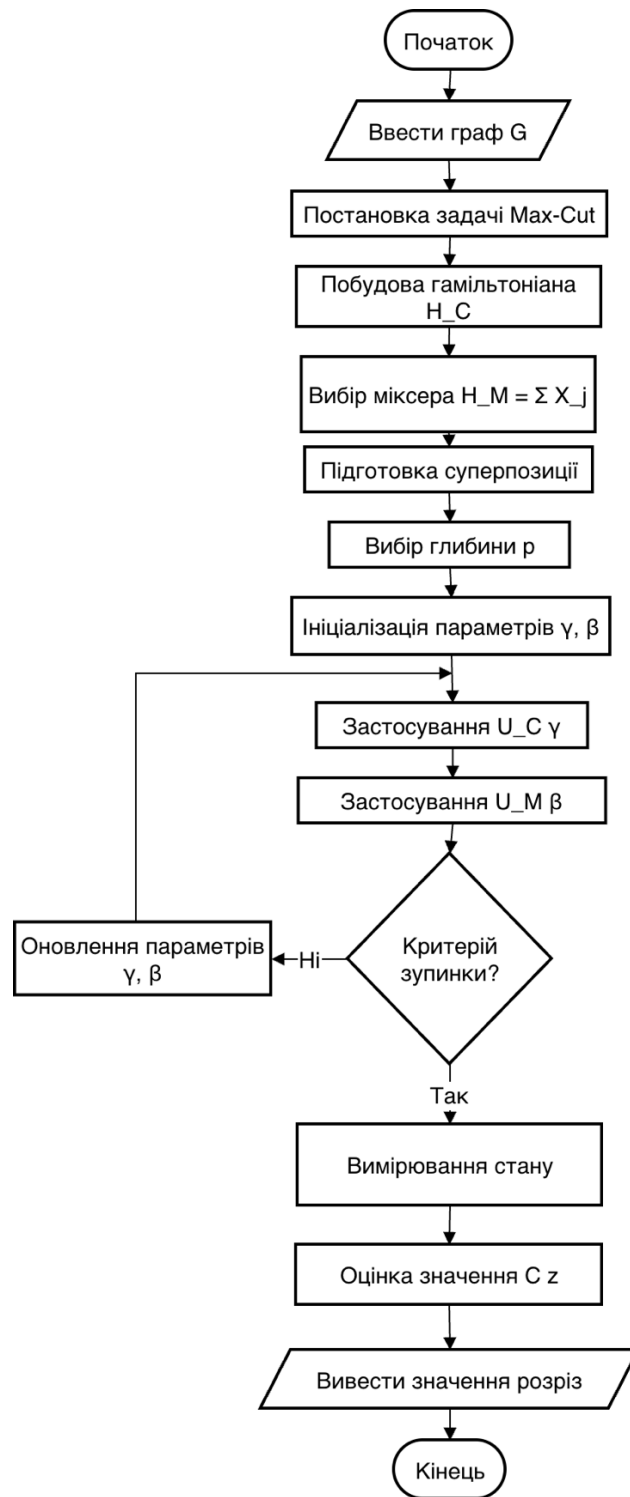


Рис. 1.29. Схема роботи алгоритму QAOA

Алгоритм розпочинається з підготовки початкового квантового стану, який являє собою рівномірну суперпозицію всіх можливих класичних

конфігурацій, що досягається шляхом застосування гейта Адамара до кожного кубіта, ініціалізованого у стані $|0\rangle$:

$$|s\rangle = H^{\otimes n} |0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{z \in \{0,1\}^n} |z\rangle. \quad (1.2)$$

Отриманий стан забезпечує представлення всього простору можливих розв'язків задачі та використовується як початковий стан у варіаційних квантових алгоритмах [71]. Наступним етапом є застосування фазового оператора $U_C(\gamma) = e^{-i\gamma H_C}$, який реалізує фазовий зсув для кожного базисного стану відповідно до значення функції вартості, визначеної гамільтоніаном H_C , при цьому параметр γ задає кут обертання та визначається у процесі класичної оптимізації [71]. Застосування цього оператора призводить до кодування значень функції вартості у фазах квантового стану, що використовується у подальшій інтерференції. Мікшерний оператор задається у вигляді $U_M(\beta) = e^{-i\beta H_M}$ та забезпечує переходи між різними конфігураціями станів, сприяючи дослідженню простору розв'язків [130,136]. У якості стандартного вибору використовується гамільтоніан $H_M = \sum_{j=1}^n X_j$, де X_j – це оператор Паулі-Х, що діє на j -й кубіт, цей оператор створює суперпозиції та заплутаність, що є необхідними для квантового прискорення, а параметр β є кутом обертання, оптимізованим класично [134]. Квантова схема QAOA для p шарів представляє собою послідовне застосування цих операторів:

$$|\gamma, \beta\rangle = U_M(\beta_p) U_C(\gamma_p) \dots U_M(\beta_1) U_C(\gamma_1) |s\rangle. \quad (1.3)$$

Після того як квантовий стан $|\gamma, \beta\rangle$ створено, наступним кроком є оцінка його очікуваного значення гамільтоніана вартості H_C :

$$\langle H_C \rangle = \langle \gamma, \beta | H_C | \gamma, \beta \rangle \quad (1.4)$$

Цю величину оцінюють шляхом багаторазових вимірювань квантового стану. Наприклад, для Max-Cut, вимірюється частота появи кожного бінарного рядка i на основі цих частот обчислюється середнє значення функції вартості $C(z)$ [77]. Класичний оптимізатор (наприклад, градієнтні методи, такі як SPSA, або безградієнтні методи, як COBYLA) отримує значення функції вартості і використовує його для оновлення параметрів γ та β з метою мінімізації (або

максимізації) $\langle H_C \rangle$, цикл триває ітеративно до тих пір, поки не буде знайдено оптимальний набір параметрів або досягнутий критерій збіжності [26].

Квантовий наближений оптимізаційний алгоритм належить до класу гібридних квантово-класичних підходів, що застосовуються для розв'язання задач комбінаторної оптимізації в умовах використання шумних квантових пристроїв проміжного масштабу (NISQ). Однією з характерних особливостей алгоритму є використання гібридної обчислювальної парадигми, яка поєднує квантові та класичні компоненти, при цьому квантова частина відповідає за генерацію параметризованих станів, а класична – за оптимізацію параметрів γ та β [80]. Така організація обчислювального процесу забезпечує можливість застосування алгоритму на сучасних квантових процесорах обмеженого розміру без необхідності використання повноцінних відмовостійких квантових систем.

Однією з характерних властивостей алгоритму QAOA є можливість його застосування до широкого класу задач комбінаторної оптимізації, зокрема задачі максимального розрізу, задач розміщення, маршрутизації та інших графових оптимізаційних задач [57]. Така властивість обумовлена способом формалізації задачі через гамільтоніан вартості, що забезпечує узагальнене представлення різних задач у межах єдиної квантової моделі. Крім того, варіаційний характер QAOA передбачає використання параметризованого квантового анзацу, параметри якого визначаються у процесі класичної оптимізації з урахуванням особливостей конкретного екземпляра задачі, що забезпечує можливість отримання наближених розв'язків із заданими властивостями.

Разом із тим, алгоритм QAOA характеризується рядом обмежень. Одним із ключових аспектів є відсутність строгих теоретичних гарантій переваги над найкращими класичними алгоритмами для всіх класів задач. Зокрема, для окремих екземплярів задачі Max-Cut результати QAOA можуть бути співставними або поступатися результатам класичного алгоритму Г'юеманса-

Вільямсона з гарантованим коефіцієнтом апроксимації [70], що ускладнює формальне обґрунтування квантового прискорення у загальному випадку.

Додаткові обмеження пов'язані з характеристиками квантових пристроїв епохи NISQ, зокрема обмеженою кількістю кубітів, обмеженнями на глибину квантових схем, а також впливом шуму та декогеренції. При збільшенні глибини схеми відбувається накопичення помилок, що може призводити до погіршення якості отриманих результатів.

Окрему складність становить задача класичної оптимізації параметрів QAOA, оскільки ландшафт функції вартості у просторі параметрів γ та β характеризується нелінійністю та наявністю значної кількості локальних мінімумів, що ускладнює знаходження оптимальних значень параметрів класичними методами оптимізації [80]. Це впливає на стабільність та відтворюваність результатів. Питання масштабованості також залишається предметом досліджень, оскільки досягнення квантового прискорення для задач великої розмірності, яке перевищує результати сучасних класичних евристичних методів, потребує подальшого розвитку як алгоритмічних підходів, так і квантового апаратного забезпечення [120].

Одним із представників класу варіаційних квантових є VQE (Variational Quantum Eigensolver), який використовується для оцінювання власних значень гамільтоніанів та має широке застосування у задачах квантової хімії та оптимізації [77]. Такий підхід є особливо актуальним в умовах сучасних квантових обчислювальних пристроїв з обмеженими ресурсами, оскільки дозволяє ефективно використовувати наявні апаратні можливості NISQ-платформ [61].

Алгоритм VQE ґрунтується на варіаційному принципі, відповідно до якого очікуване значення енергії параметризованого квантового стану визначає верхню межу енергії основного стану відповідного гамільтоніана. У межах гібридної обчислювальної процедури квантова частина алгоритму забезпечує підготовку параметризованого квантового стану (анзацу), тоді як класичний оптимізатор виконує мінімізацію очікуваного значення

гамільтоніана шляхом коригування варіаційних параметрів [77]. Така схема застосовується для апроксимації основних станів квантових систем та використовується у задачах молекулярного моделювання і квантової хімії [14]. Завдяки гібридній архітектурі алгоритм VQE може застосовуватися в умовах шумних квантових пристроїв, що обумовлено обмеженою глибиною квантових схем та використанням класичної оптимізації.

Подальший розвиток цього підходу привів до появи адаптивних модифікацій, зокрема алгоритму ADAPT-VQE, у якому структура анзацу формується динамічно шляхом ітеративного додавання операторів із заздалегідь визначеного пулу [18]. У межах ADAPT-VQE на кожній ітерації здійснюється оцінювання градієнтів енергії за відповідними операторами, після чого до анзацу додається оператор, що забезпечує найбільший внесок у зменшення значення енергії, що дозволяє обмежувати кількість варіаційних параметрів і контролювати глибину квантової схеми [62]. Порівняно з фіксованими анзацами, зокрема UCCSD, такі підходи забезпечують досягнення заданої точності із використанням меншої кількості параметрів і двокубітних операцій [14]. Існують різні модифікації ADAPT-VQE, зокрема fermionic-ADAPT-VQE та qubit-ADAPT-VQE, які відрізняються вибором пулу операторів та особливостями побудови анзацу, що впливає на складність квантової схеми та процедуру оптимізації [16]. Крім того, результати вимірювань, отримані під час оцінки енергії, можуть бути використані повторно у процесі оптимізації без істотного збільшення кількості вимірювань. За досягнення заданого рівня точності кількість двокубітних CNOT-вентилів у сформованих схемах наближається до мінімально необхідної, що визначає придатність таких підходів для реалізації на сучасних квантових пристроях [129].

Одним із напрямів розвитку квантових оптимізаційних алгоритмів є використання підходів так званого теплового старту (warm-start), які передбачають використання результатів, отриманих класичними методами оптимізації, для ініціалізації квантових алгоритмів [130]. Такий підхід

базується на поєднанні класичних і квантових обчислювальних процедур та застосовується в умовах обмежених ресурсів сучасних квантових пристроїв. Використання warm-start у поєднанні з алгоритмом QAOA пов'язане з можливістю зменшення кількості ітерацій класичної оптимізації та покращення збіжності алгоритму, зокрема для схем малої глибини [127]. Практична реалізація теплового старту передбачає використання наближених або оптимальних розв'язків, отриманих класичними методами, зокрема методами напіввизначеного програмування, для формування початкового квантового стану або ініціалізації параметрів варіаційної схеми.

Такий підхід забезпечує перенесення властивостей класичних розв'язків, включаючи їх апроксимаційні характеристики, у квантову процедуру оптимізації [32]. Крім того, застосування warm-start пов'язане зі зменшенням вимог до квантових обчислювальних ресурсів, що має значення для реалізації алгоритмів на NISQ-пристроях [130]. Зокрема, для задачі максимального розрізу можливе використання векторних представлень, отриманих у результаті релаксації задачі напіввизначеного програмування, для побудови початкового квантового стану, який використовується як початкова конфігурація у подальшій варіаційній оптимізації [32].

Незважаючи на розвиток квантових технологій та зростання кількості досліджень у цій галузі, сучасні квантові обчислювальні системи характеризуються рядом обмежень, які впливають на можливості їх практичного застосування, зокрема для розв'язання задач комбінаторної оптимізації, таких як Max-Cut (рис. 1.30). Поточний етап розвитку квантових обчислень описується як NISQ-ера, що визначається обмеженою кількістю кубітів та наявністю шуму у квантових процесорах [60]. Такі обмеження ускладнюють реалізацію квантових схем великої глибини та обумовлюють необхідність використання гібридних квантово-класичних підходів для отримання практично значущих результатів.

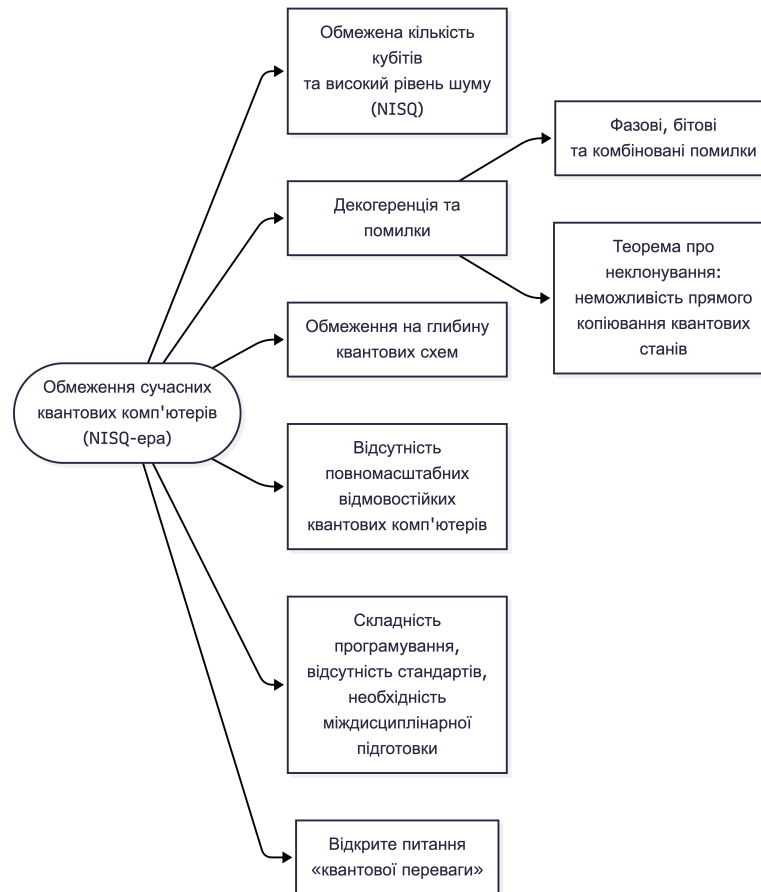


Рис. 1.30. Схема обмежень сучасних квантових комп'ютерів

Квантові стани характеризуються високою чутливістю до зовнішніх впливів, що призводить до виникнення декогеренції та накопичення помилок різної природи, зокрема фазових і бітових, а також їх комбінацій [38]. Ускладнення процесів корекції помилок пов'язане з теоремою про не клонування, яка унеможливорює пряме копіювання квантових станів для їх відновлення [38], що обмежує надійність виконання складних квантових алгоритмів на сучасному апаратному забезпеченні.

Додаткові обмеження пов'язані з кількістю доступних кубітів і допустимою глибиною квантових схем, які можуть бути реалізовані на сучасних квантових пристроях [67]. Зокрема, реалізація алгоритму факторизації, запропонованого Шором П. (Shor P.), для чисел практично значущого розміру потребує використання відмовостійких квантових комп'ютерів, які на сьогодні залишаються предметом досліджень [60]. Водночас галузі квантового машинного навчання та оптимізації

характеризуються відсутністю уніфікованих підходів і високою складністю програмування, що зумовлює необхідність міждисциплінарної підготовки для коректного формулювання задач та їх відображення у квантових обчислювальних моделях [67].

Питання досягнення практичної квантової переваги залишається предметом досліджень [65]. Хоча квантові алгоритми демонструють потенційні переваги для окремих класів задач, у більшості практичних сценаріїв оптимізовані класичні алгоритми забезпечують вищу продуктивність порівняно з реалізаціями на сучасних гейтових квантових системах, за окремими винятками [35]. У зв'язку з цим подальші дослідження спрямовані на аналіз впливу розмірності задач, шумових факторів та умов експлуатації з метою оцінювання ефективності квантових методів у прикладних сценаріях.

Висновки до першого розділу

Здійснено аналіз сучасного стану квантових технологій та методів розв'язання задачі максимального розрізу з використанням класичних, квантових і квантово-гібридних підходів, що сформуло цілісне уявлення про еволюцію обчислювальних парадигм у контексті розв'язання NP-складних задач комбінаторної оптимізації.

Визначено, що задача Max-Cut є фундаментальною NP-складною проблемою з широким спектром практичних застосувань у задачах маршрутизації, кластеризації, аналізу мереж та оптимізації ресурсів. Аналіз класичних алгоритмів розв'язання задачі дозволив встановити, що точні методи, незважаючи на гарантію знаходження глобально оптимального розв'язку, характеризуються експоненційною часовою складністю та є обчислювально непридатними для графів великої розмірності. Зв'язано, що апроксимаційні підходи забезпечують теоретично обґрунтовані межі якості рішень, проте їх практичне застосування ускладнюється значними обчислювальними витратами, тоді як евристичні методи, попри кращу

масштабованість, не гарантують стабільної якості результатів і часто демонструють чутливість до вибору початкових умов.

Здійснено аналіз сучасного розвитку квантових технологій як потенційної альтернативи класичним методам оптимізації. Розглянуто базові принципи квантових обчислень, зокрема роль однокубітних та багатокубітних квантових вентилів у формуванні універсальних наборів операцій і створенні квантової заплутаності як ключового ресурсу квантових алгоритмів. Детальне дослідження квантових гейтів дозволило сформувати основу для подальшого аналізу структур квантових схем, що застосовуються у квантових алгоритмах.

Проаналізовано квантово-гібридні алгоритми, зокрема Quantum Approximate Optimization Algorithm у варіантах зі стандартною ініціалізацією та з «теплим стартом», а також алгоритм Variational Quantum Eigensolver. Показано, що дані підходи поєднують переваги квантових обчислень у формуванні складних квантових станів із гнучкістю класичних методів оптимізації параметрів, що робить їх перспективними для реалізації на сучасних квантових пристроях. Встановлено, що ефективність таких алгоритмів суттєво залежить від топології квантових схем, вибору параметризації та властивостей ландшафту функції вартості.

Проаналізовано обмеження сучасних квантових технологій, що зумовлені перебуванням квантових обчислювальних пристроїв у NISQ-ері. Визначено, що обмежена кількість кубітів, шум, декогеренція та помилки двокубітних операцій істотно ускладнюють реалізацію глибоких квантових схем і знижують стабільність варіаційних алгоритмів. Виявлено, що додатковою проблемою є складна структура параметричного простору, що може призводити до застрягання класичних оптимізаторів у локальних мінімумах і, як наслідок, до отримання субоптимальних рішень.

Таким чином, результати першого розділу свідчать про те, що існуючі класичні, квантові та квантово-гібридні підходи не забезпечують повною мірою ефективного та масштабованого розв'язання задачі Max-Cut для великих і складних графів. Виявлені обмеження та виклики формують наукове підґрунтя

для подальших досліджень, спрямованих на розробку нових квантово-гібридних методів і алгоритмічних структур, здатних підвищити якість рішень та адаптуватися до реальних апаратних обмежень сучасних квантових систем.

Матеріали розділу 1 подано в таких публікаціях автора [2–4, 116, 117].

РОЗДІЛ 2. МЕТОДИ ТА МОДЕЛІ РОЗРОБКИ КВАНТОВО-ГІБРИДНОГО АЛГОРИТМУ

2.1. Моделювання квантово-гібридного алгоритму QASPA

Сучасні квантові обчислювальні системи представляють складні багаторівневі програмно-апаратні комплекси, у яких квантовий процесор є лише одним із компонентів загальної обчислювальної інфраструктури. На відміну від класичних обчислювальних систем, квантові обчислювальні пристрої не функціонують як автономні системи, а потребують постійної взаємодії з класичними комп'ютерами, які забезпечують керування квантовими операціями, обробку результатів вимірювань, оптимізацію параметрів та взаємодію з користувачем. Унаслідок цього навіть на рівні апаратної архітектури квантові обчислення реалізуються у квантово-класичній парадигмі.

Характерною особливістю сучасних квантових обчислювальних систем є наявність класичних обчислювальних компонентів не лише на рівні зовнішнього керування, але й безпосередньо в межах квантового обчислювального блоку. Такі компоненти забезпечують синхронізацію керуючих імпульсів, реалізацію квантових гейтів, калібрування пристрою, обробку похибок та зчитування результатів вимірювань. У зв'язку з цим квантовий комп'ютер доцільно розглядати як ієрархічну систему, у якій квантові та класичні підсистеми інтегровані та функціонують у межах єдиного обчислювального процесу (рис. 2.1).

У контексті розв'язання задач комбінаторної оптимізації, зокрема задачі максимального розрізу графа, зазначені архітектурні особливості безпосередньо впливають на організацію обчислювального процесу. Квантово-гібридні алгоритми, до яких належить квантовий алгоритм наближеного оптимізування (QAOA), реалізуються у вигляді ітераційної взаємодії між користувачем, класичною обчислювальною підсистемою та квантовим процесором. При цьому користувач не взаємодіє з квантовим комп'ютером безпосередньо, а здійснює керування обчислювальним процесом через класичне програмне забезпечення,

яке формує квантову схему, забезпечує запуск квантових обчислень та виконує обробку отриманих результатів.

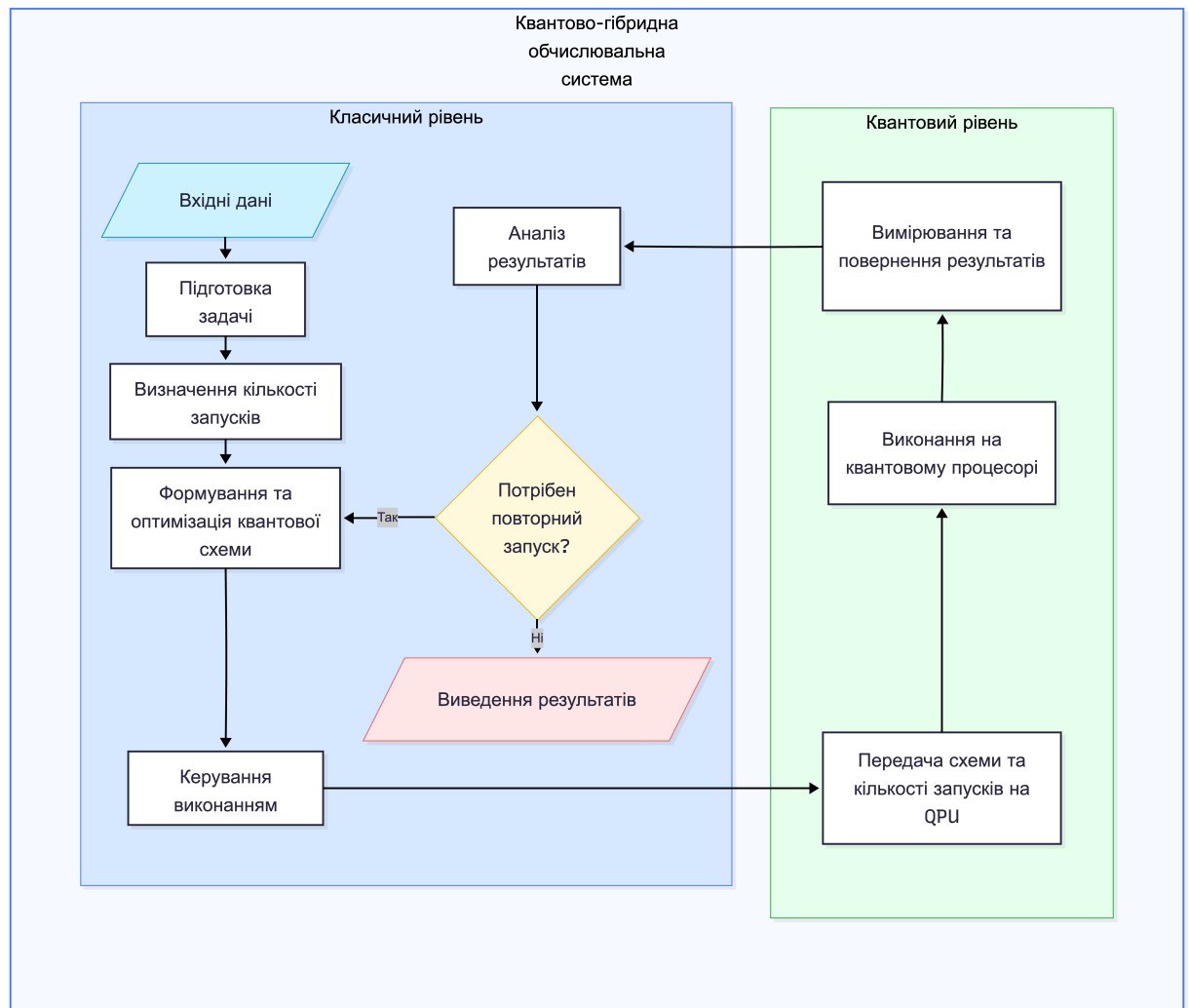


Рис. 2.1. Архітектура квантового комп'ютера

Процес взаємодії користувача з квантово-гібридною системою на прикладі алгоритму QAOA характеризується послідовністю етапів, що передбачає задання параметрів алгоритму, формування параметризованої квантової схеми, виконання квантових обчислень, вимірювання квантового стану та подальшу класичну оптимізацію параметрів на основі отриманих результатів. Для відображення зазначеної взаємодії використовується схема взаємодії алгоритму QAOA (рис. 2.2), що відображає ролі користувача, класичної та квантової підсистем, а також ітераційний характер квантово-гібридного процесу оптимізації.



Рис. 2.2. Діаграма послідовності алгоритму QAOA

Узагальнюючи наведене, моделювання квантово-гібридного процесу розв’язання задачі Max-Cut потребує комплексного підходу, що враховує як алгоритмічні особливості квантових методів, так і архітектурні обмеження сучасних квантових обчислювальних систем. Попри відносну простоту формалізації квантово-гібридних алгоритмів, ітераційна схема взаємодії квантової та класичної підсистем, що використовується в алгоритмі QAOA, характеризується рядом обмежень. Ключовою особливістю QAOA є необхідність багаторазового виконання квантових схем для кожної ітерації класичного оптимізатора з метою оцінювання значення цільової функції. Такий підхід пов’язаний із зростанням часових витрат, що обумовлено як тривалістю виконання квантових обчислень, так і накладними витратами на обмін даними між класичною та квантовою обчислювальними підсистемами.

У практичних реалізаціях алгоритму QAOA кожна ітерація оптимізації передбачає формування параметризованої квантової схеми, її виконання на квантовому процесорі або симуляторі, багаторазове вимірювання квантового стану з метою отримання статистично значущих результатів, а також подальшу класичну обробку отриманих даних. За наявності шумів, похибок зчитування та обмеженого часу когерентності кубітів кількість необхідних вимірювань зростає, що призводить до збільшення загального часу виконання алгоритму.

Окремим аспектом ітераційної схеми QAOA є затримки, пов'язані з передаванням даних між класичним комп'ютером та квантовим пристроєм. У хмарних квантових сервісах такі затримки можуть бути співставними або перевищувати час виконання квантових обчислень, що впливає на загальну ефективність алгоритму. Крім того, багаторазові виклики квантового процесора пов'язані зі збільшенням обчислювальних витрат та обмеженнями масштабування для задач великої розмірності.

Зазначені особливості свідчать про те, що ітераційна квантово-гібридна схема, яка лежить в основі класичного варіанта QAOA, не завжди є ефективною з точки зору часових і ресурсних витрат. Це обумовлює актуальність дослідження альтернативних підходів до організації квантово-гібридного процесу, спрямованих на зменшення кількості взаємодій між квантовою та класичною підсистемами.

Одним із напрямів удосконалення квантово-гібридних алгоритмів є переосмислення багатоітераційної схеми оптимізації, характерної для QAOA, та перехід до підходів, орієнтованих на виконання квантового обчислення з обмеженою кількістю запусків або в рамках одного запуску, що пов'язано зі зменшенням накладних витрат, зумовлених багаторазовою взаємодією між квантовою та класичною підсистемами, та скороченням загального часу виконання алгоритму в умовах обмежених ресурсів сучасних квантових пристроїв.

У межах даного дисертаційного дослідження розглядається квантово-гібридний підхід, реалізований у вигляді алгоритму QASPA (Quantum Approximate Shift-Phase Algorithm), який базується на одноітераційній схемі організації обчислювального процесу. На відміну від класичного QAOA, у запропонованому підході параметри квантової схеми визначаються заздалегідь на класичному рівні та використовуються для формування квантового обчислення без подальшої ітераційної корекції.

Такий підхід дозволяє усунути необхідність багаторазових циклів квантово-класичної взаємодії та звести процес виконання алгоритму до одного

квантового запуску. Основні принципи побудови та реалізації алгоритму QASPA розглядаються далі у тексті роботи.

Для представлення організації квантово-гібридного процесу з використанням алгоритму QASPA наведено схему взаємодії (рис. 2.3), яка відображає послідовність дій користувача, класичної обчислювальної підсистеми та квантового процесора в межах одноітераційного підходу. Аналіз обмежень класичної багатоітераційної схеми квантово-гібридних алгоритмів зумовив перехід до модифікованої організації квантово-класичної взаємодії, що передбачає зменшення кількості ітерацій до однієї.

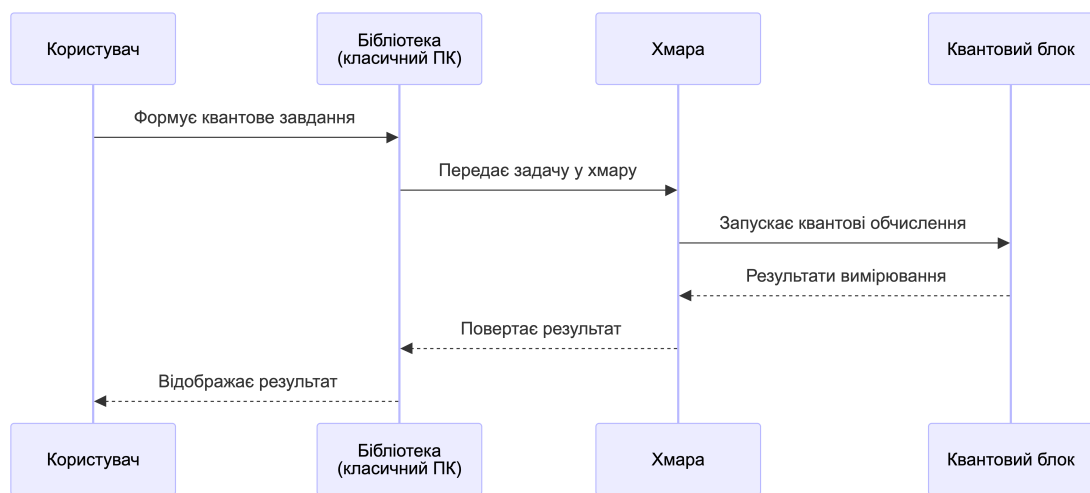


Рис. 2.3. Схема взаємодії у роботі алгоритму QASPA

Ключовою ідеєю запропонованого підходу є попереднє визначення параметрів квантової схеми на класичному рівні з урахуванням структури графа задачі Max-Cut, зокрема зв'язків між вершинами та ваг ребер. Така організація обчислювального процесу забезпечує виконання квантового обчислення без подальшої ітераційної корекції параметрів.

Для реалізації одноітераційного квантово-гібридного алгоритму QASPA сформовано узагальнену структурно-функціональну блок-схему, подану на рис. 2.4-2.5. Вона відображає повний цикл обробки даних, включаючи класичну підготовку задачі, побудову параметризованої квантової схеми, її виконання на квантовому процесорі та подальший класичний аналіз результатів.

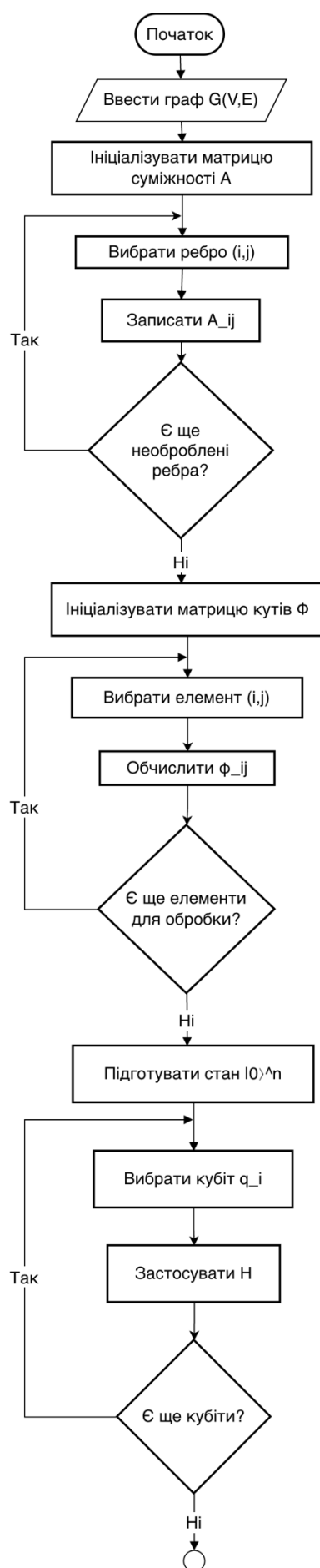


Рис. 2.4. Блок-схема роботи квантово-гібридного алгоритму QASPA частина 1

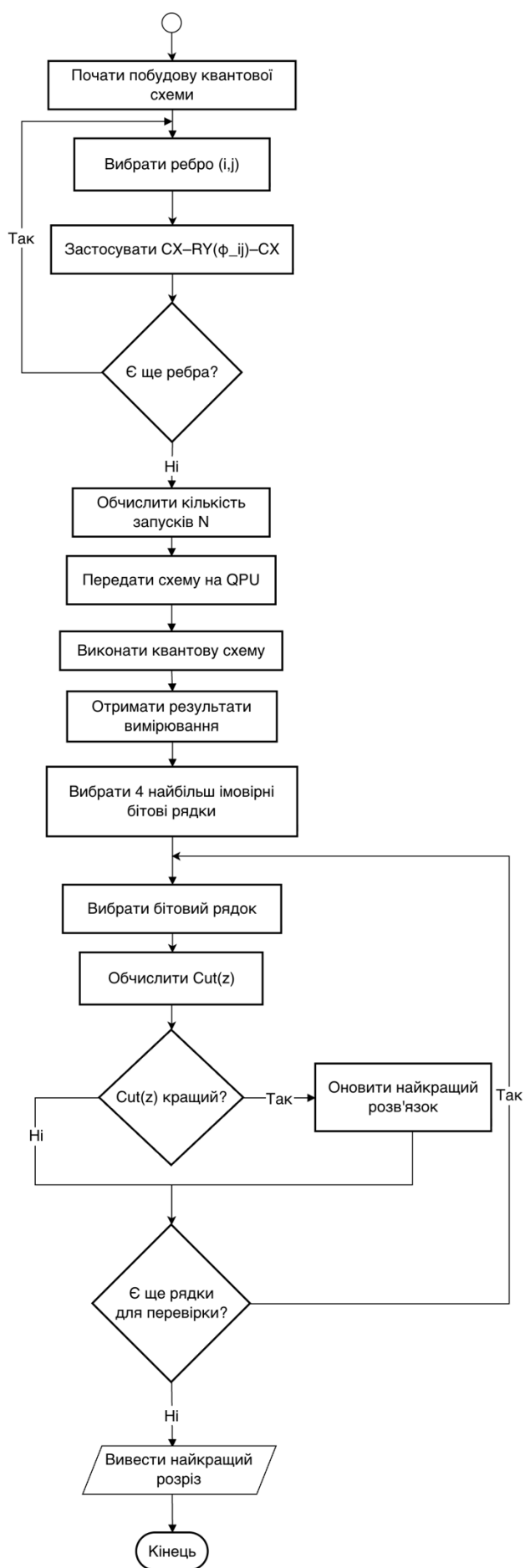


Рис. 2.5. Блок-схема роботи квантово-гібридного алгоритму QASPA частина 2

На етапі попередньої обробки вхідний граф перетворюється у матрицю суміжності, на основі якої формується матриця кутів, що використовується для параметризації квантової схеми. Після підготовки початкового стану та побудови квантового оператора здійснюється виконання заданої кількості запусків на квантовому процесорі.

Отримані результати у вигляді розподілу ймовірностей бітових рядків обробляються класичною системою аналізу, яка визначає найбільш імовірні кандидати та обчислює значення цільової функції, що дозволяє отримати фінальний розв'язок задачі.

З огляду на це, дії класичного комп'ютера можна виразити через математичний опис задачі Max-Cut та її подання у матричній формі. На початковому етапі задача максимального розрізу формулюється на неорієнтованому зваженому графі

$$G = (V, E, W), \quad (2.1)$$

де $V = \{v_1, \dots, v_n\}$ – множина вершин графа розміру $n = |V|$, що безпосередньо відповідає кількості кубітів у квантовій частині алгоритму. Множина ребер

$$E \subseteq \{(i, j) \mid i < j, j \in \{1, \dots, n\}\} \quad (2.2)$$

визначає топологію взаємодії між вершинами, а вагова функція

$$W: E \rightarrow \mathbb{R}_{\geq 0} \quad (2.3)$$

присвоює кожному ребру $(i, j) \in E$ невід'ємне дійсне число w_{ij} . За відсутності ребра між вершинами i та j відповідна вага вважається рівною нулю.

Метою задачі Max-Cut є знаходження такого розбиття множини вершин V на дві взаємовиключні та вичерпні підмножини V_1 і V_2 , що

$$V = V_1 \cup V_2, V_1 \cap V_2 = \emptyset, \quad (2.4)$$

при якому сумарна вага ребер, що з'єднують вершини з різних підмножин, є максимальною:

$$C(V_1, V_2) = \sum_{(i,j) \in E} w_{ij} \cdot \mathbb{1}[v_i \in V_1, v_j \in V_2 \text{ або навпаки}], \quad (2.5)$$

де $\mathbb{1}[\cdot]$ – індикаторна функція, що набуває значення 1, якщо умова виконується, і 0 – в іншому випадку.

Для подальшої параметризації квантової схеми графова структура задачі переводиться у матричне подання у вигляді симетричної матриці суміжності

$$A \in \mathbb{R}^{n \times n}, \quad (2.6)$$

елементи якої визначаються як $a_{ij} = w_{ij}$, для $i \neq j$ та $a_{ii} = 0$. Таке подання є зручним для класичної обробки та водночас забезпечує прямий зв'язок між топологією графа та структурою квантової схеми. З метою коректного та уніфікованого відображення ваг ребер у кути квантових обертань, що зберігають унітарність операторів, у загальному випадку вводиться нормалізована матриця ваг

$$\tilde{A} = [\tilde{w}_{ij}], \quad (2.7)$$

елементи якої належать заданому інтервалу допустимих значень. Нормалізація є критично важливою для зважених графів, оскільки забезпечує коректне призначення кутів обертання квантових вентилів. У класичному варіанті задачі Max-Cut, де $w_{ij} \in \{0,1\}$, нормування не є необхідним, однак запропонований підхід легко узагальнюється на випадок зважених графів. Після перетворення графу у матричний вигляд подальша робота класичної частини алгоритму зосереджується на побудові матриці кутів та параметризації масштабування кількості запусків на квантовій частині, моделі та методи яких описані у підрозділі 2.2.

Після чого, на основі матриці кутів формується параметризована квантова схема (рис. 2.6), структура якої безпосередньо відображає топологію графа задачі Max-Cut. Побудова квантової схеми, математична модель якої описана у підрозділі 2.3, для розв'язання задачі максимального розрізу здійснюється відповідно до структури вхідного неорієнтованого графа де кількість кубітів дорівнює кількості вершин графа. На початковому етапі формується квантовий регістр $n = |V|$ кубітів, при цьому кожному кубіту ставиться у відповідність окрема вершина графа.

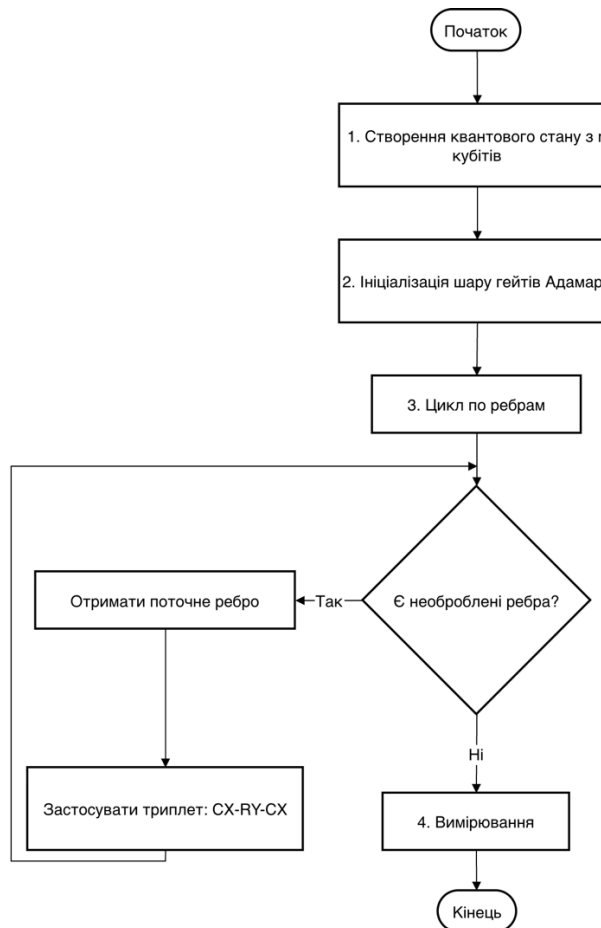


Рис. 2.6. Схема кодування ребер у кубіти

Усі кубіти ініціалізуються у базисному стані, що забезпечує коректний старт квантової еволюції системи. Далі до кожного кубіта застосовується шар гейтів Адамара, внаслідок чого формується рівноймовірна суперпозиція всіх можливих бінарних конфігурацій. Такий підхід дозволяє одночасно представити весь простір потенційних розрізів графа та створює початковий простір пошуку для подальшої оптимізації. Наступний етап передбачає кодування структури графа у квантову схему. Для цього здійснюється послідовний обхід множини ребер графа, і для кожного ребра $(i, j) \in E$ застосовується комбінація гейтів CX-RY-CX. Застосування такого триплету вентилів забезпечує контрольовану модифікацію амплітуд квантових станів з урахуванням взаємозв'язків між відповідними вершинами графа. У результаті квантова система еволюціонує таким чином, що інформація про топологію графа інтегрується у розподіл імовірностей вимірюваних станів. Після завершення обробки всіх ребер виконується вимірювання квантового регістру в

обчислювальному базисі. Отримані бітові послідовності інтерпретуються як можливі варіанти розрізу графа та передаються на класичний етап аналізу для обчислення значення цільової функції та вибору найкращого розв'язку.

Наступним етапом являється запуск квантової схеми на квантовому комп'ютері, який у свою чергу, здійснює багаторазовий запуск підготовленої квантової схеми, використовуючи метод масштабування кількості запусків, який описаний у підрозділі 2.2, виконуючи квантову еволюцію та вимірювання квантового стану. Результатом цього етапу є розподіл імовірностей бітових рядків, що відповідають можливим розбиттям множини вершин графа.

Подальша обробка результатів здійснюється класичною системою аналізу (рис. 2.7), яка декодує отримані бітові конфігурації, обчислює значення цільової функції по (2.5) та формує кінцевий результат розв'язання задачі максимального розрізу. Таким чином, класична система аналізу завершує квантово-гібридний цикл, не ініціюючи повторного запуску квантового обчислення.

Запропонований алгоритм для розв'язання задачі Max-Cut відрізняється від класичних квантово-гібридних алгоритмів, зокрема QAOA, використанням одноітераційної схеми квантових обчислень. Така організація обчислювального процесу пов'язана зі зменшенням загального часу виконання алгоритму та скороченням накладних витрат, зумовлених багаторазовою взаємодією між класичною та квантовою підсистемами. З урахуванням обмежень сучасних квантових пристроїв, зокрема обмеженого часу когерентності та наявності шуму, зменшення кількості квантових запусків є важливим фактором ефективності квантово-гібридних алгоритмів. У межах запропонованого підходу класична частина відповідає за підготовку задачі та визначення параметрів квантової схеми, квантова частина – за еволюцію стану з використанням параметризованих унітарних операторів, а класична обробка результатів забезпечує інтерпретацію отриманих розв'язків. Така організація квантово-гібридного процесу дозволяє застосовувати запропонований підхід для наближеного розв'язання задачі Max-Cut як на квантових симуляторах, так і на реальних квантових пристроях.

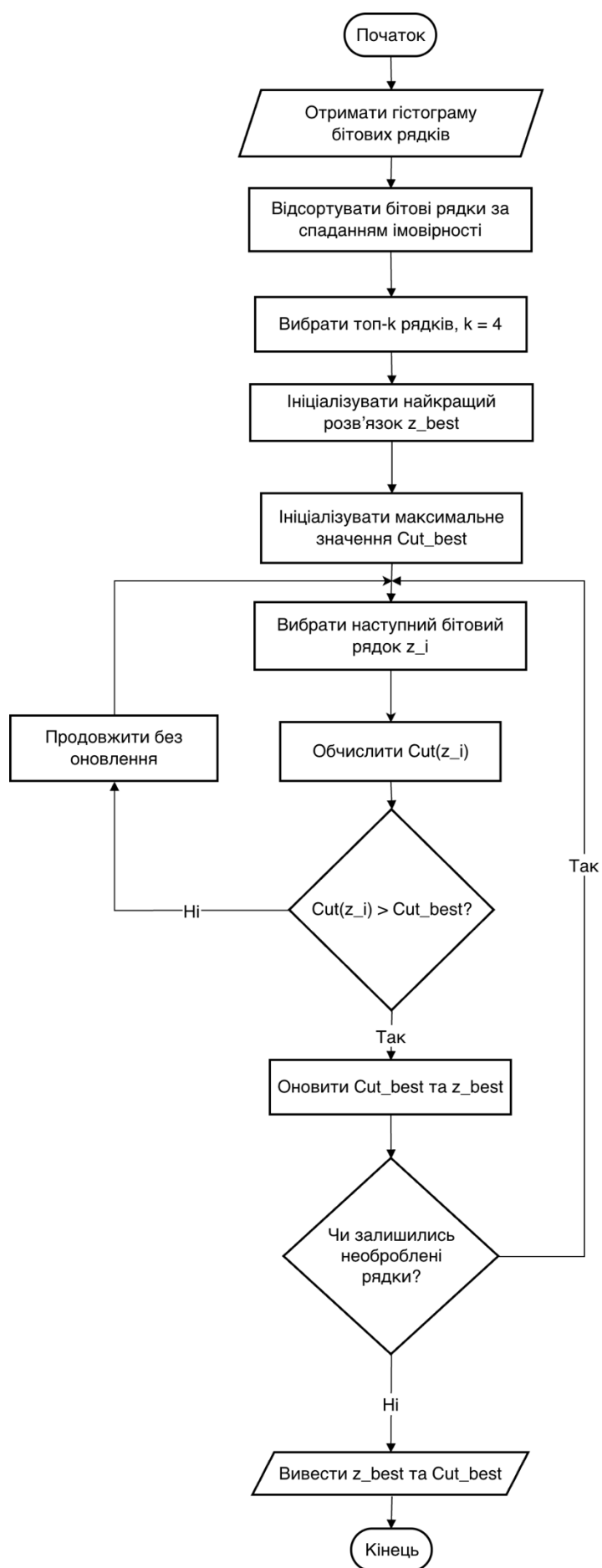


Рис. 2.7. Блок-схема роботи класичної системи аналізу алгоритму QASPA

Таким чином, квантово-гібридний алгоритм QASPA визначає чітку послідовність інформаційних перетворень між класичною та квантовою підсистемами, де ключову роль відіграє етап класичної підготовки задачі. Саме на цьому етапі здійснюється формалізація задачі максимального розрізу у вигляді структури даних, придатної для подальшого використання у квантовій схемі.

2.2. Методи генерації кутів та масштабування кількості запусків алгоритму QASPA

Наступним етапом класичної частини алгоритму є генерація матриці кутів

$$\Phi = [\phi_{ij}], \quad (2.8)$$

яка безпосередньо визначає параметри однокубітних та контрольованих квантових обертань у квантовій схемі (рис. 2.8).

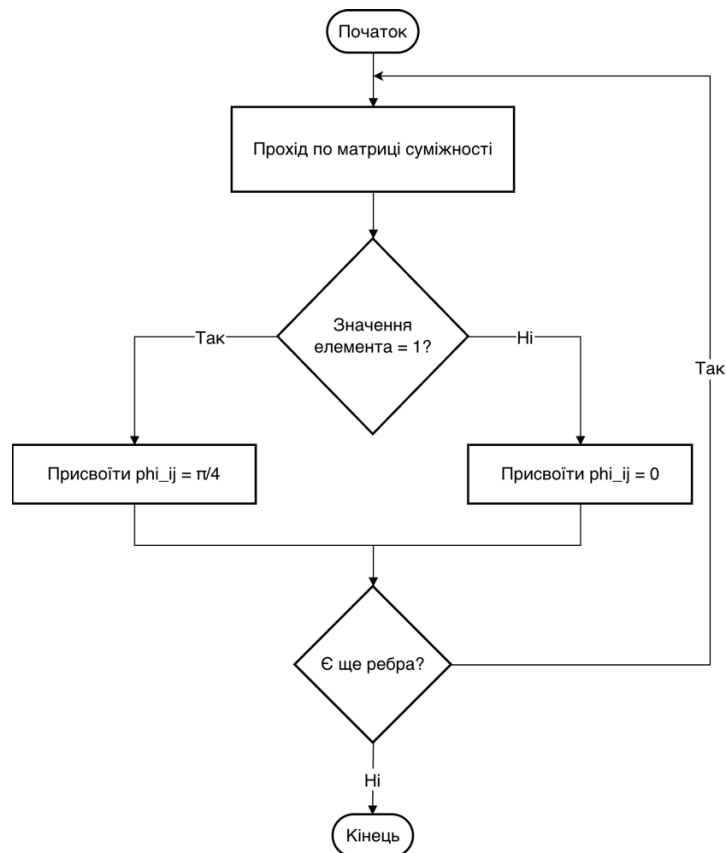


Рис. 2.8. Процес генерації кутів у алгоритмі QASPA

Матриця кутів задає параметризацію для квантової схеми, а саме перетворення зв'язків графа у зв'язки між квантовими бітами. У базовому варіанті моделі, узгодженому з геометрією обертань на сфері Блоха, кути визначаються за правилом:

$$\phi_{ij} = \begin{cases} \frac{\pi}{4}, & w_{ij} > 0, \\ 0, & w_{ij} = 0, \end{cases} \quad (2.9)$$

а у випадку нормалізованої матриці ваг – за формулою:

$$\phi_{ij} = \frac{\pi}{4} \cdot \tilde{w}_{ij}. \quad (2.10)$$

Застосування вагової модуляції дозволяє безпосередньо інтегрувати структуру графа у параметри квантових обертань, надаючи алгоритму гнучкості та можливості адаптації до гетерогенних графів. Вибір фіксованого значення $\phi_{ij} = \pi/4$ для всіх наявних ребер є збалансованим компромісом, який був визначений шляхом систематичного експериментального аналізу на множині графів різної структури та щільності. Початково у коді було задано множину потенційних оптимальних значень кута, а також параметри серії запусків:

```
angles = [np.pi/2, np.pi/3, np.pi/4]
```

```
num_trials = 10
```

```
graph_sizes = list(range(3, 15))
```

```
SUCCESS_THRESHOLD = 0.9
```

```
base_seed = 42
```

Таким чином, розглядалися значення $\theta \in \{\pi/2, \pi/3, \pi/4\}$, а експеримент проводився для графів із кількістю вершин $n \in [3; 14]$. Для кожного розміру виконувалося десять повторів. Введення порогового значення $SUCCESS_THRESHOLD = 0.9$ у даному дослідженні має як методологічне, так і практичне обґрунтування та спрямоване на перехід від аналізу лише середнього значення відношення наближення до оцінювання стійкості та надійності роботи алгоритму в різних структурних умовах. По-перше, використання виключно середнього значення $\rho = C_{qaspa}/C_{opt}$ може маскувати наявність нестабільних запусків. Наприклад, середнє значення $\rho \approx 0.95$ може

бути отримане як за умови рівномірно високих результатів (усі $\rho \in [0.93; 0.97]$), так і внаслідок поєднання дуже добрих і суттєво гірших розв'язків (наприклад, частина запусків із $\rho \approx 1.0$, а частина з $\rho \approx 0.75$). З точки зору практичного застосування алгоритму для задач комбінаторної оптимізації така варіативність є небажаною, оскільки вона знижує передбачуваність якості результату. Введення умови $\rho \geq 0.9$ дозволяє інтерпретувати кожен запуск як «прийнятний» або «неприйнятний» з позиції прикладної ефективності та перейти до аналізу частки надійних розв'язків. По-друге, значення 0.9 відповідає 10% відхиленню від глобального оптимуму, що є розумним компромісом між строгістю критерію та практичною досяжністю для гібридних квантових алгоритмів на задачах NP-складності. У класичних дослідженнях апроксимаційних алгоритмів для задачі Max-Cut відхилення в межах 5-15% від оптимуму розглядається як високоякісний результат для евристичних або варіаційних підходів. Таким чином, поріг 0.9 задає достатньо строгий критерій, який не занижує вимоги до алгоритму, але й не є надмірно жорстким (як, наприклад, $\rho \geq 0.99$, що могло б некоректно дискваліфікувати стабільно хороші, але не абсолютно оптимальні розв'язки). Також було введено змінну `base_seed = 42` яка забезпечила відтворюваність генерації випадкових графів. Формування тестових задач було реалізовано у функції генерації графів:

```
def generate_graphs(graph_size, trial_num):
    graphs = {}

    G_complete = nx.complete_graph(graph_size)
    graphs['Complete'] = G_complete

    G_cycle = nx.cycle_graph(graph_size)
    graphs['Cycle'] = G_cycle

    G_er_03 = nx.erdos_renyi_graph(graph_size, 0.3, seed=base_seed)
    if G_er_03.number_of_edges() == 0:
        G_er_03.add_edge(0, 1)
    graphs['Erdos-Renyi (p=0.3)'] = G_er_03

    G_er_05 = nx.erdos_renyi_graph(graph_size, 0.5, seed=base_seed)
```

```

if G_er_05.number_of_edges() == 0:
    G_er_05.add_edge(0, 1)
graphs['Erdos-Renyi (p=0.5)'] = G_er_05

G_er_07 = nx.erdos_renyi_graph(graph_size, 0.7, seed=base_seed)
if G_er_07.number_of_edges() == 0:
    G_er_07.add_edge(0, 1)
graphs['Erdos-Renyi (p=0.7)'] = G_er_07

return graphs

```

У кожному повторі для заданого розміру n формувалася набір графів п'яти типів: повний граф як граничний випадок максимальної щільності, цикл як регулярна розріджена структура та три реалізації випадкових графів Ердеша-Реньї з імовірностями $p = 0.3, 0.5, 0.7$. Також була передбачена перевірка на відсутність ребер у випадковому графі гарантує коректність постановки задачі Max-Cut та уникнення вироджених випадків з нульовим оптимумом. Використання фіксованого seed означає, що для кожного розміру n структура випадкового графа залишається сталою при зміні кута θ , що дозволяє коректно інтерпретувати відмінності результатів саме як наслідок варіювання параметра angle. Основний цикл дослідження організовано наступним чином:

```

for graph_size in graph_sizes:
    for trial in range(num_trials):
        # Generate different graph types
        graphs = generate_graphs(
            graph_size, angle_idx*1000 + graph_size*100 + trial)

        for graph_type, G in graphs.items():
            # Initialize storage for this graph type if needed
            if graph_type not in all_results[angle_key]:
                all_results[angle_key][graph_type] = {
                    'correct': 0,
                    'total': 0,
                    'ratios': [],
                    'success_rate': 0,
                    'avg_ratio': 0
                }

            # Run brute force to get optimal cut
            try:

```

```

        brute_force_cut, _ = brute_force(G)
except Exception as e:
    print(
        f' Error: brute force failed for {graph_type} (size {graph_size}):
{str(e)})

    continue

# Run QASPA with the current angle
try:
    qaspa_cut, _, _ = run_qaspa_maxcut_local(
        G, debug=False, angle=angle)

    all_results[angle_key][graph_type]['total'] += 1

    # Calculate approximation ratio
    approx_ratio = qaspa_cut / brute_force_cut if brute_force_cut > 0 else 0
    all_results[angle_key][graph_type]['ratios'].append(
        approx_ratio)

    # Check if result meets 90% success threshold
    if approx_ratio >= SUCCESS_THRESHOLD:
        all_results[angle_key][graph_type]['correct'] += 1

except Exception as e:
    print(
        f' Error in {graph_type} (size {graph_size}, trial {trial+1}): {str(e)})
    continue

```

У межах цього вкладеного циклу для кожного значення кута θ послідовно перебиралися всі розміри графів та повтори, після чого для кожного типу графа виконувалися два ключові обчислення – визначення еталонного оптимуму та запуск QASPA. Спочатку застосовувався точний алгоритм повного перебору, а отримане значення `brute_force_cut` інтерпретувалося як C_{opt} – точне значення максимального розрізу. Далі для того самого графа виконувався запуск квантово-гібридного алгоритму з фіксованим параметром кута, результат виконання якого(`qaspa_cut`) відповідає значенню C_{qaspa} , отриманому алгоритмом QASPA-MaxCut. Таким чином, кожна пара викликів `brute force` – QASPA формує одну експериментальну точку для фіксованої трійки параметрів (θ , n , тип графа). Якість наближення (`approx_ratio`) оцінювалася через відношення $\rho = C_{qaspa}/C_{opt}$. Далі

здійснювався підрахунок успішних запусків та накопичення значень для обчислення середнього показника (all_results). Після завершення повного циклу для кожного кута обчислювалися агреговані характеристики – частка успішних запусків (success_rate) та середнє значення ρ (avg_ratio). Завершальний етап дослідження полягав у визначенні глобально оптимального кута, що максимізує середній показник успішності по всіх типах графів:

$\text{best_angle} = \max(\text{angle_avg_success}, \text{key}=\text{angle_avg_success.get})$

Експериментальні результати показали (рис. 2.9), що значення $\pi/4$ забезпечує максимальний середній показник успішності, який становить 100%, а середнє відношення наближення дорівнює 0.9931.

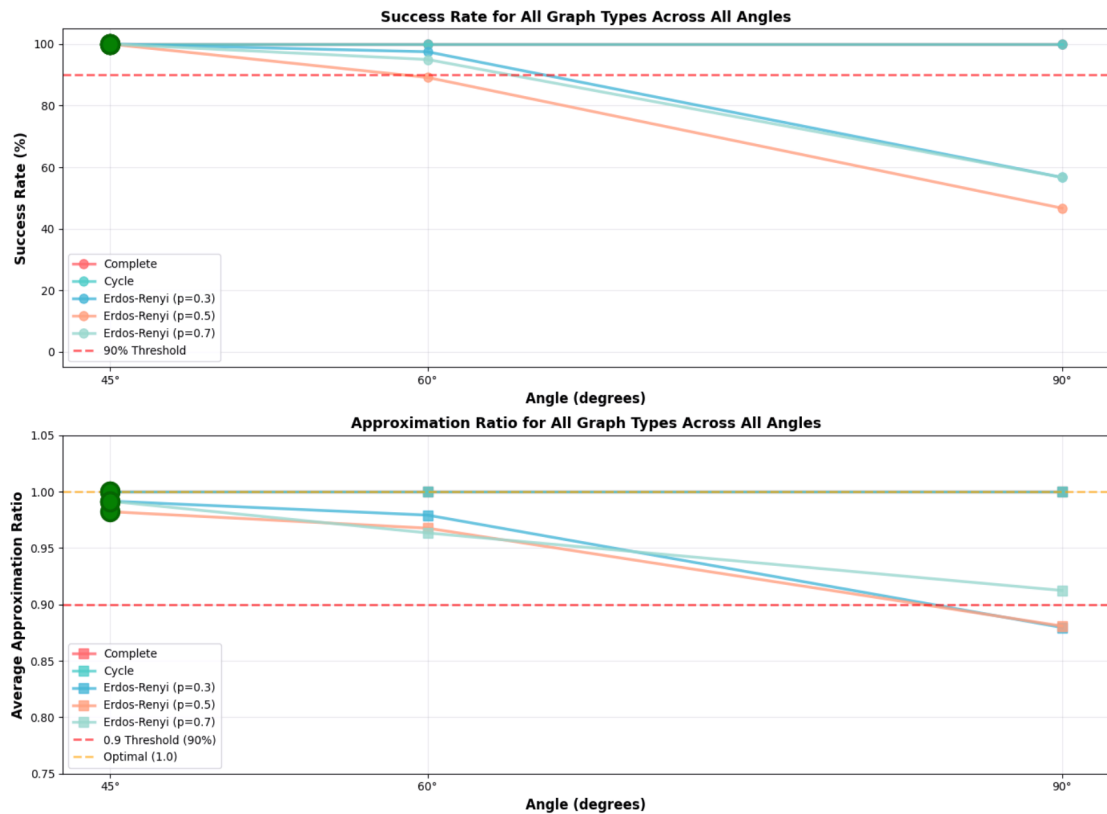


Рис. 2.9. Результати експерименту по підбору оптимального кута

Для повних та циклічних графів алгоритм відтворює оптимум у всіх запусках ($\rho = 1.0$), тоді як для випадкових графів середні значення ρ залишаються стабільно високими (понад 0.98), причому жоден запуск не опускається нижче порогового значення 0.9. Натомість для $\pi/3$ та $\pi/2$ спостерігається помітна деградація показників саме на випадкових графах

середньої та високої щільності, що свідчить про підвищену чутливість алгоритму до значення параметра при надмірному фазовому зсуві. Отже, проведене параметричне дослідження з використанням повного перебору як еталонного розв'язку, багаторазових запусків QASPA та узагальнення результатів за критеріями частки успішних розв'язків і середнього відношення наближення дозволило обґрунтовано обрати значення кута $\pi/4$ як універсальний параметр для подальших експериментів (рис. 2.10).

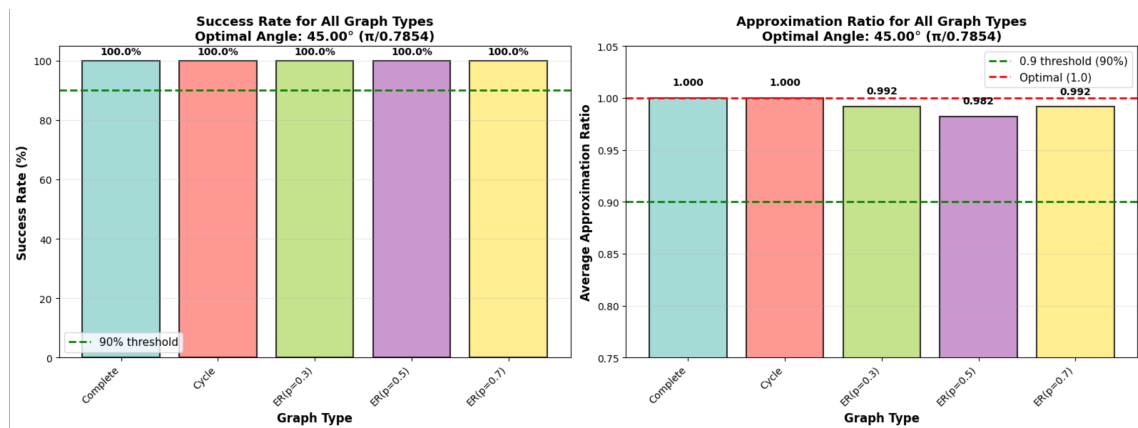


Рис. 2.10. Результати експерименту для кута $\pi/4$

Описана вище процедура формування матриці кутів Φ визначає параметризацію квантової схеми та характер квантової еволюції у межах одноітераційного алгоритму QASPA. Проте коректна інтерпретація результатів виконання квантової частини алгоритму залежить не лише від структури квантового анзацу, але й від статистичних характеристик процесу вимірювання. У цьому контексті ключовим параметром квантово-гібридного процесу є кількість квантових запусків N , яка безпосередньо впливає на надійність виділення оптимальних розв'язків задачі Max-Cut. Під час виконання квантової частини алгоритму результати вимірювань формуються за наявності шумів та апаратних похибок, властивих сучасним квантовим пристроям. Джерела цих похибок є багатофакторними та залежать як від фізичної реалізації квантового процесора, так і від глибини квантової схеми, типів використаних вентилів та характеристик окремих кубітів.

За недостатньої кількості запусків квантового обчислення отриманий розподіл результатів має високий рівень шуму, що ускладнює виділення бітових конфігурацій, які відповідають оптимальним або наближеним розв'язкам задачі максимального розрізу. Було розглянуто граф розмірністю 16 для розробки для проведення експерименту по оптимізації кількості запусків на квантовій частині, який зображений на рис. 2.11.

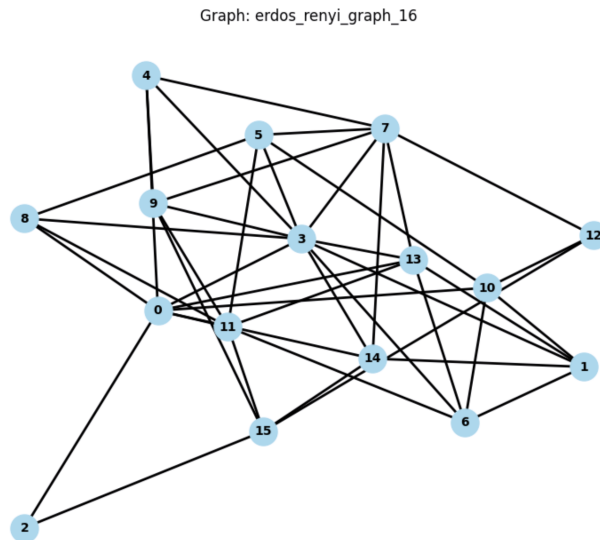


Рис. 2.11. Граф з 16 вершинами, що використовувався для візуалізації

Для вибраного графу був виконаний запуск з кількістю shorts=32 (рис. 2.12), у результаті виконання якого було отримано результати які не давали оптимального результату, не оброблювали всіх можливих комбінацій вершин, а потенційні розв'язки були розчинені в шумі. З огляду на це, кількість запусків N повинна бути достатньою для формування помітних піків у гістограмі вимірювань, які відповідають найбільш імовірним бітовим рядкам. Запропонований алгоритм QASPA за рахунок квантових інтерференційних ефектів підсилює оптимальні розв'язки задачі Max-Cut, однак ефективність такого підсилення безпосередньо залежить від статистичної вибірки результатів.

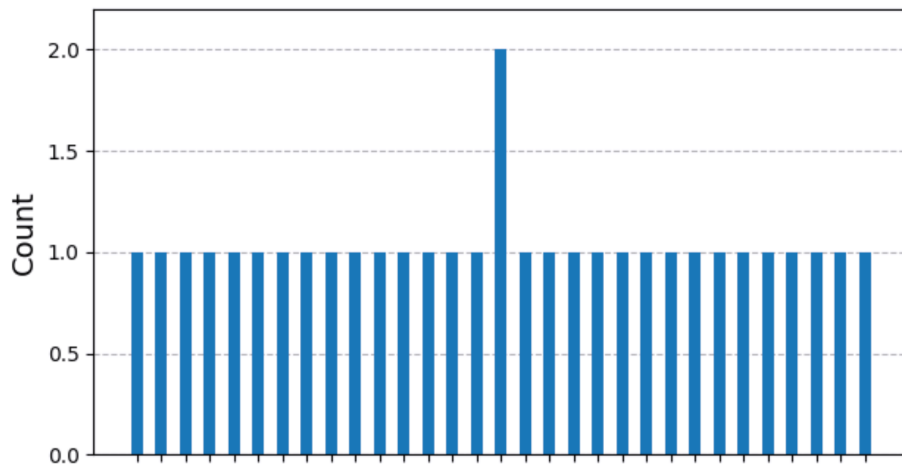


Рис. 2.12. Запуск алгоритму з shorts=32 для графа розмірністю 16

Для малих графів відношення кількості корисних результатів до шумового фону є прийнятним навіть за фіксованої кількості запусків, тоді як зі збільшенням розмірності задачі ситуація суттєво змінюється. При зростанні кількості вершин графа n простір можливих бітових конфігурацій $S = 2^n$ збільшується експоненціально, що призводить до розсіювання результатів вимірювань по значно більшій кількості станів. У такому випадку навіть правильні результати квантового обчислення можуть статистично «розчинятися» у шумовому фоні, якщо кількість запусків залишається недостатньою. Звідси випливає принциповий висновок: для збереження надійності ідентифікації оптимальних розв'язків кількість запусків N повинна зростати разом із розмірністю задачі n . На рис. 2.13 наведено приклад гістограми вимірювань, у якій помітні піки відповідають потенційним розв'язкам задачі максимального розрізу для графа з рис. 2.11, де кількість запусків було збільшено відносно розмірності графа. Метою методу масштабування кількості запусків є підбір такого значення N , яке з високою ймовірністю забезпечує появу цих піків над рівнем шуму для заданої розмірності задачі.

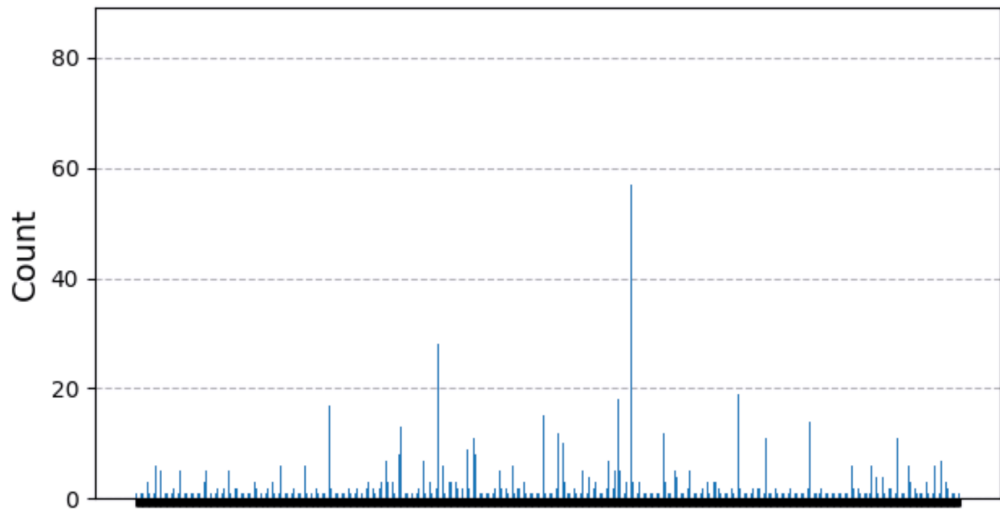


Рис. 2.13. Гістограма результатів виконання квантової частини

За умов помірного квантового підсилення та достатньої кількості запусків (рис. 2.14) очікується, що кожен оптимальний або квазіоптимальний бітовий рядок матиме статистично значуще представлення у вибірці результатів. Якщо оптимальних розв'язків існує декілька, їх сумарний внесок формує характерну «шапку» гістограми, в якій один або декілька домінантних піків чітко виділяються на тлі решти конфігурацій. Водночас за недостатнього підсилення або малої кількості запусків частота появи таких рядків може бути меншою за одиницю, що унеможливило їх виявлення без суттєвого збільшення N .



Рис. 2.14. Відношення кількості запусків до вершин графа

Емпіричні дослідження показали, що для малих графів із кількістю вершин $n \leq 8$ достатньою є кількість запусків $N = 1024$, тоді як для більших графів із $n \geq 17$ надійне виділення піків досягається за $N \geq 5012$. Для забезпечення плавного переходу між цими режимами без необхідності додаткового ручного налаштування під час експериментів у роботі запропоновано афінно-експоненційну апроксимацію залежності кількості запусків від розмірності задачі (рис. 2.15). Нехай функція $N^*(n)$ описує рекомендовану кількість запусків для графа з n вершинами та має вигляд

$$N^*(n) = \lceil \alpha \cdot 2^n + \beta \rceil, \quad (2.11)$$

де коефіцієнти α та β визначаються з умов проходження через дві контрольні точки: $N(8) = 1024, N(17) = 5012$. Розв'язуючи відповідну систему, отримуємо

$$\alpha = \frac{5012 - 1024}{2^{17} - 2^8} = \frac{3988}{130816} \approx 0.03047, \quad (2.12)$$

$$\beta = 1024 - \alpha \cdot 2^8 \approx 1016.2. \quad (2.13)$$

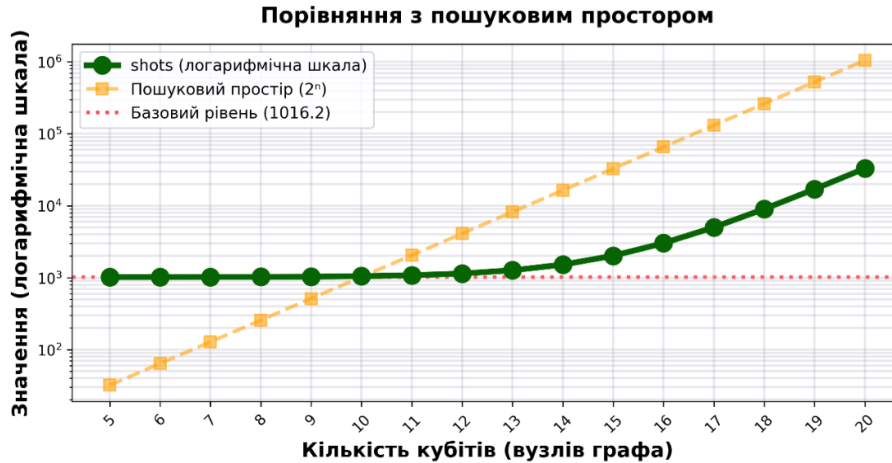


Рис. 2.15. Відношення просторового пошуку до кількості запусків

У результаті остаточною формула для визначення кількості запусків квантової частини алгоритму набуває вигляду

$$N^*(n) = \lceil 0.03047 \cdot 2^n + 1016.2 \rceil. \quad (2.14)$$

Запропонована гладка залежність (рис. 2.16) дозволяє адаптивно масштабувати експериментальні дослідження, забезпечуючи достатню

статистичну значущість результатів вимірювань навіть для задач великої розмірності, де простір можливих розв'язків зростає експоненціально.

2.3. Метод побудови квантової схеми алгоритму QASPA

Побудоване на попередньому етапі матричне подання задачі Max-Cut та визначення параметрів квантової схеми дозволяють перейти до строгого опису квантової еволюції, яка реалізує запропонований квантово-гібридний алгоритм. Квантова частина процесу розглядається як послідовність унітарних перетворень над обчислювальним регістром, що кодує бінарні конфігурації розбиття вершин графа. Для наочного та відтворюваного аналізу подальший виклад здійснюється на конкретному прикладі графа з трьома вершинами, у якому множина ребер визначається як

$$E = \{(0,1), (1,2), (2,0)\}. \quad (2.15)$$

Така структура поєднує циклічну топологію з центральною вершиною та створює показовий приклад для демонстрації формування запутаності між кубітами та впливу топології графа на квантову динаміку системи. Квантовий обчислювальний регістр містить $n = |V|$ кубітів, кожен з яких відповідає окремій вершині графа та кодує її належність до однієї з частин розрізу. Початковий стан регістру ініціалізується у вигляді $|0\rangle^{\otimes n}$, що відповідає детермінованій бітовій конфігурації та не несе інформації про структуру задачі. Для забезпечення рівномірного представлення всіх можливих бітових конфігурацій на першому етапі виконується підготовка глобальної суперпозиції, після чого до квантового стану може бути внесена інформація про топологію графа та параметри задачі.

Початковий стан обчислювального регістру $|0\rangle^{\otimes n}$ переводиться у глобальну суперпозицію застосуванням шару гейтів Адамара (рис. 2.16). Вибір саме гейта Адамара зумовлений його геометричною дією на сфері Блоха: вектор стану кожного кубіта переноситься з північного полюса на екватор, що забезпечує рівні амплітуди $|0\rangle$ та $|1\rangle$ і створює передумову квантового паралелізму – рівномірне представлення всіх бітових конфігурацій

до «внесення» структури графа. За відсутності цього шару подальші керовані обертання діяли б на розріджений стан і не формували б необхідного інтерференційного зсуву.

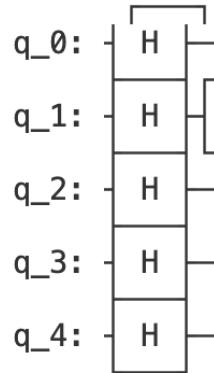


Рис. 2.16. Гейти Адамара на квантовій схемі

Оператор Адамара H визначається матрицею

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad (2.16)$$

і реалізує відображення базисних станів одного кубіта у рівномірну суперпозицію. Для багатокубітної системи застосовується тензорний добуток $H^{\otimes n}$, який діє незалежно на кожен кубіт обчислювального регістру. Застосування цього оператора до початкового стану призводить до наступного перетворення:

$$|\psi_1\rangle = H^{\otimes n} |0\rangle^{\otimes n} = \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}} \right)^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle. \quad (2.17)$$

Отриманий квантовий стан є рівномірною суперпозицією всіх можливих бітових рядків довжини n . Кожному базисному стану $|x\rangle$, де

$$x = (x_0, x_1, \dots, x_{n-1}) \in \{0,1\}^n, \quad (2.18)$$

відповідає однакова амплітуда

$$\alpha_x = \frac{1}{\sqrt{2^n}}, \quad (2.19)$$

а ймовірність отримання цього стану в результаті вимірювання в обчислювальному базисі дорівнює

$$P(x) = |\alpha_x|^2 = \frac{1}{2^n}. \quad (2.20)$$

Таким чином, після шару Адамара квантова система знаходиться у максимально симетричному стані, у якому всі можливі бітові конфігурації представлені з однаковою ймовірністю. На цьому етапі квантовий алгоритм не містить жодної інформації про структуру графа або цільову функцію задачі, а лише формує рівномірний простір пошуку.

Розглянемо застосування цієї формалізації до конкретної задачі Max-Cut для графа E оголошеного у 2.15, множини вершин якого $V = \{0,1,2\}$. У цьому випадку кількість кубітів квантового регістру дорівнює кількості вершин графа, тобто $n = 3$. Початковий стан обчислювального регістру задається як

$$|\psi_0\rangle = |0\rangle^{\otimes 3} = |000\rangle. \quad (2.21)$$

У стандартному впорядкуванні обчислювального базису

$$|000\rangle, |001\rangle, |010\rangle, |011\rangle, |100\rangle, |101\rangle, |110\rangle, |111\rangle, \quad (2.22)$$

цей стан має векторне подання розмірності 8×1 , у якому ненульовою є лише перша компонента. Після застосування шару вентилів Адамара отримуємо стан

$$|\psi_1\rangle = H^{\otimes 3} |000\rangle = \frac{1}{\sqrt{8}} \sum_{x \in \{0,1\}^3} |x\rangle, \quad (2.23)$$

тобто рівномірну суперпозицію всіх $2^5 = 32$ можливих бітових конфігурацій.

У векторному поданні цей стан має вигляд

$$|\psi_1\rangle = \frac{1}{\sqrt{8}} (1 \ 1 \ 1 \ \dots \ 1)^T, \quad (2.24)$$

де кожна компонента відповідає окремому бітовому рядку

$$x = (x_0, x_1, x_2, x_3, x_4) \in \{0,1\}^3. \quad (2.25)$$

Кожен такий рядок інтерпретується як можливе розбиття множини вершин графа на дві підмножини, де значення x_i визначає належність вершини i до однієї з частин розрізу. Після шару Адамара всі розбиття є рівноймовірними, а квантова система не віддає переваги жодній конфігурації. Отже, шар Адамара формує початковий стан квантового пошуку, який охоплює весь простір можливих розв'язків задачі Max-Cut.

Після формування рівномірної суперпозиції за допомогою шару вентилів Адамара наступним етапом квантового алгоритму є внесення

інформації про структуру задачі безпосередньо у квантовий стан. Для цього використовується параметризований двокубітний унітарний блок (рис. 2.17), який асоціюється з окремим ребром графа та реалізується композицією вентилів типу $CX - R_y - CX$. Такий блок забезпечує контрольоване змішування амплітуд базисних станів залежно від взаємного розташування відповідної пари вершин у бітовій конфігурації.

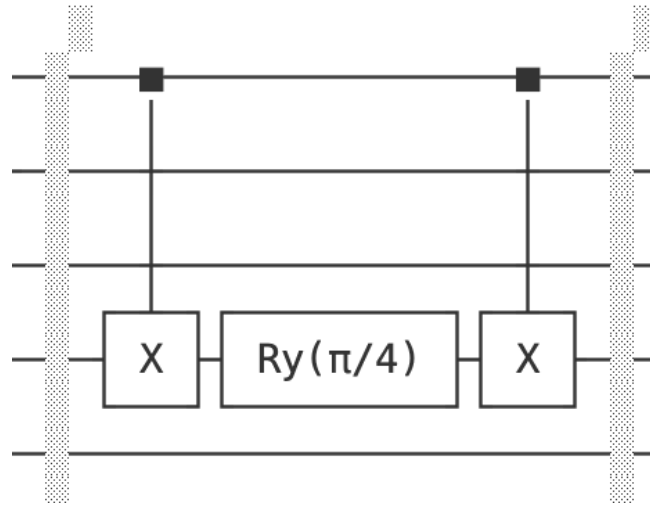


Рис. 2.17. Триплет $CX-R_y-CX$

Для формального опису впливу окремого ребра на квантовий стан розглянемо довільну n -кубітну квантову систему та фіксовану пару кубітів i, j . Відповідний унітарний оператор, що реалізує зв'язок між цими кубітами, задається композицією двох керованих інверсій та параметризованого однокубітного повороту:

$$U_{i \rightarrow j}(\theta_{ij}) = \widetilde{CX}_{i \rightarrow j} \tilde{R}_y^{(j)}(\theta_{ij}) \widetilde{CX}_{i \rightarrow j}, \quad (2.26)$$

де θ_{ij} – параметр повороту, асоційований з ребром (i, j) , який береться з побудованої матриці кутів Φ . Таким чином, повний квантовий стан системи після виконання алгоритму визначається виразом:

$$|\psi_{final}\rangle = \left(\prod_{(i,j) \in E} U_{i \rightarrow j} \right) H^{\otimes n} |0\rangle^{\otimes n}, \quad (2.27)$$

де символ $\prod_{(i,j) \in E} U_{i \rightarrow j}$ позначає послідовну композицію операторів $U_{i \rightarrow j}$ для всіх ребер графа $G = (V, E)$, а n це кількість вершин. Оператор $\widetilde{CX}_{i \rightarrow j} \in \mathbb{C}^{2^n \times 2^n}$

є багатокубітним вбудуванням вентиля CNOT з керуючим кубітом i та цільовим кубітом j , сам CNOT можна виразити матрицею:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}, \quad (2.28)$$

що являється двокубітним оператором, тобто матриця коректна лише в просторі $\mathbb{C}^2 \otimes \mathbb{C}^2$, що не підходить для використання в n -кубітному просторі, для цього матрицю потрібно доповнити таким чином, щоб її можна було застосувати для в'язого квантового вектора у просторі станів $\mathbb{C}^{2^n \times 2^n}$, але за умови, що дії будуть проведені тільки з цільовими кубітами, тому для графа E (2.15) з розмірністю $n = 3$, $\widetilde{CX}_{0 \rightarrow 1}$ можна виразити матрицею 8 на 8:

$$\widetilde{CX}_{0 \rightarrow 1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}. \quad (2.29)$$

Для побудови математичних обчислень була розроблена програма мовою програмування python, а саме за формування матриць CX відповідає функція CX:

```
def CX(n: int, control: int, target: int) -> np.ndarray:
    dim = 2**n
    U = np.zeros((dim, dim), dtype=complex)

    for x in range(dim):
        bits = list(map(int, bitstring(x, n))) # біти у порядку [q_{n-1}, ..., q_0]
        # Перетворює індекс кубіта (0..n-1) на позицію в цьому списку:
        # q_k відповідає bits[n-1-k]
        cpos = n - 1 - control
        tpos = n - 1 - target

        y_bits = bits.copy()
        if bits[cpos] == 1:
            y_bits[tpos] ^= 1
```

```
y = int("".join(map(str, y_bits)), 2)
U[y, x] = 1.0
```

```
print(f"Побудовано CX({control} -> {target}) для n={n} кубітів.")
print(U.real)
```

```
return U
```

Однокубітний поворот $R_y(\theta)$ задається експоненціальною формою

$$R_y(\theta) = \exp(-i \frac{\theta}{2} Y), \text{ де } Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad (2.30)$$

і в обчислювальному базисі має матричне подання

$$R_y(\theta) = \begin{pmatrix} \cos \frac{\theta}{2} & -\sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{pmatrix} = \begin{pmatrix} c & -s \\ s & c \end{pmatrix}, \text{ де } c = \cos \frac{\theta}{2}, \text{ а } s = \sin \frac{\theta}{2}. \quad (2.31)$$

Для n -кубітного регістру відповідний оператор будується за допомогою тензорного добутку з одиничними матрицями:

$$\tilde{R}_y^{(j)}(\theta) = I_2^{\otimes(n-1-j)} \otimes R_y(\theta) \otimes I_2^{\otimes j}, \quad (2.32)$$

де I_2 – одиничний оператор розмірності 2×2 , $R_y(\theta)$ – стандартний однокубітний оператор повороту навколо осі Y , а символ $\otimes$ позначає тензорний добуток операторів. Для обчислення оператора R_Y було розроблено функції `Ry` та `embed_1q`, яка вбудовує однокубітний оператор R_Y у n -кубітний регістр:

```
def Ry(theta: float) -> np.ndarray:
    c = cos(theta/2)
    s = sin(theta/2)
    return np.array([[c, -s],
                     [s, c]], dtype=complex)

def embed_1q(U: np.ndarray, n: int, j: int) -> np.ndarray:
    ops = []
    for k in range(n-1, -1, -1):
        ops.append(U if k == j else I2)
    return kron_n(ops)
```

Така форма запису забезпечує коректне розширення однокубітного оператора до повного гільбертового простору $(\mathbb{C}^2)^{\otimes n}$ та гарантує, що поворот застосовується виключно до кубіта з індексом j , тоді як стани всіх інших

кубітів залишаються незмінними. Для графа N матричне відображення $\tilde{R}_y^{(1)}(\theta)$ буде виглядати:

$$\tilde{R}_y^{(1)}(\theta) = I_2 \otimes R_y(\theta) \otimes I_2 = \begin{pmatrix} c & 0 & -s & 0 & 0 & 0 & 0 & 0 \\ 0 & c & 0 & -s & 0 & 0 & 0 & 0 \\ s & 0 & c & 0 & 0 & 0 & 0 & 0 \\ 0 & s & 0 & c & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & c & 0 & -s & 0 \\ 0 & 0 & 0 & 0 & 0 & c & 0 & -s \\ 0 & 0 & 0 & 0 & s & 0 & c & 0 \\ 0 & 0 & 0 & 0 & 0 & s & 0 & c \end{pmatrix}, (2.33)$$

де s та c взяті з (2.30). Відповідно повний математичний опис триплету, зв'язку між першим та другим ребром заданого графа N , виглядає наступним чином:

$$U_{0 \rightarrow 1}(\theta) = \begin{pmatrix} c & 0 & -s & 0 & 0 & 0 & 0 & 0 \\ 0 & c & 0 & s & 0 & 0 & 0 & 0 \\ s & 0 & c & 0 & 0 & 0 & 0 & 0 \\ 0 & -s & 0 & c & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & c & 0 & -s & 0 \\ 0 & 0 & 0 & 0 & 0 & c & 0 & s \\ 0 & 0 & 0 & 0 & s & 0 & c & 0 \\ 0 & 0 & 0 & 0 & 0 & -s & 0 & c \end{pmatrix} (2.34)$$

Дія оператора $U_{i \rightarrow j}(\theta_{ij})$ на сформований гейтами Адамара стан суперпозиції $|\psi_1\rangle$ описується множенням відповідного вектора амплітуд на унітарну матрицю розмірності $2^n \times 2^n$:

$$|\psi_2\rangle = U_{i \rightarrow j}(\theta_{ij}) |\psi_1\rangle (2.35)$$

Завдяки блоково-діагональній структурі оператора $\tilde{R}_y^{(j)}(\theta)$ його дія зводиться до попарного лінійного перетворення амплітуд для кожної фіксованої конфігурації всіх бітів, окрім x_j . З урахуванням двох перестановочних операторів $\tilde{C}\tilde{X}_{i \rightarrow j}$ повний блок $CX - R_y - CX$ реалізує кероване обертання, яке залежить від значення біта x_i . Таким чином, амплітуди базисних станів, для яких $x_i \neq x_j$, піддаються іншому змішуванню, ніж амплітуди станів з $x_i = x_j$, що безпосередньо відповідає ідеї підсилення конфігурацій, у яких відповідне ребро є «перерізанним». Для повного обчислення математичної моделі графа трикутника, був реалізований програмний код, що значно спростив проведення обчислень:

```
E_triangle = [(0, 1), (0, 2), (1, 2)]
```

```
def cut_value(bitstr: str, edges) -> int:
```

```
    n = len(bitstr)
```

```
    def x_v(v: int) -> int:
```

```
        return int(bitstr[n - 1 - v])
```

```
    C = 0
```

```
    for (u, v) in edges:
```

```
        C += 1 if (x_v(u) ^ x_v(v)) == 1 else 0
```

```
    return C
```

```
n = 3
```

```
theta = pi / 4
```

```
# Початковий стан та шар Адамара
```

```
psi0 = ket0(n)
```

```
Hn = kron_n([H] * n)
```

```
psi1 = Hn @ psi0
```

```
# Визначає оператори ребер  $U_{\{i \rightarrow j\}}(\theta) = CX * Ry^j(\theta) * CX$ 
```

```
def U_edge(n: int, i: int, j: int, theta: float) -> np.ndarray:
```

```
    CXij = CX(n, i, j)
```

```
    Ryj = embed_1q(Ry(theta), n, j)
```

```
    return CXij @ Ryj @ CXij
```

```
# Застосовує ребра трикутника у виборі порядку: (0,1) потім (0,2) потім (1,2)
```

```
U01 = U_edge(n, 0, 1, theta)
```

```
psi2 = U01 @ psi1
```

```
U02 = U_edge(n, 0, 2, theta)
```

```
psi3 = U02 @ psi2
```

```
U12 = U_edge(n, 1, 2, theta)
```

```
psi4 = U12 @ psi3
```

Для ілюстрації прикладного обчислювання значення розглянемо приклад графа N та ребро $(0,1)$, що використовується для побудови матриці 2.32. Нехай параметр повороту дорівнює $\theta_{01} = \pi/4$. Тоді обчислюючи значення c та s з (2.30) отримаємо:

$$c = \cos \frac{\theta}{2} = \cos \frac{\pi}{8} = \sqrt{\frac{1 + \cos(\pi/4)}{2}} = \frac{1}{2} \sqrt{2 + \sqrt{2}}, \quad (2.36)$$

$$s = \sin \frac{\theta}{2} = \sin \frac{\pi}{8} = \sqrt{\frac{1 - \cos(\pi/4)}{2}} = \frac{1}{2} \sqrt{2 - \sqrt{2}}. \quad (2.37)$$

Після застосування блоку $U_{0 \rightarrow 1}(\pi/4)$ одержуємо кінцевий вектор стану

$$|\psi_2\rangle = U_{0 \rightarrow 1}(\theta) |\psi_1\rangle = \frac{1}{\sqrt{8}} \begin{pmatrix} c - s \\ c + s \\ c + s \\ c - s \\ c - s \\ c + s \\ c + s \\ c - s \end{pmatrix} \approx \begin{pmatrix} 0.19134 \\ 0.46194 \\ 0.46194 \\ 0.19134 \\ 0.19134 \\ 0.46194 \\ 0.46194 \\ 0.19134 \end{pmatrix}. \quad (2.38)$$

Чисельно це відповідає підсиленню амплітуд для тих бітових рядків, у яких значення бітів x_0 та x_1 різні, та зменшенню амплітуд для конфігурацій з $x_0 = x_1$. Отже, блок $CX - R_y - CX$ виконує елементарний крок перенесення структурної інформації про окреме ребро у квантовий стан, створюючи інтерференційний зсув, необхідний для подальшого наближеного розв'язання задачі Max-Cut. Далі виконаємо два аналогічні перетворення для ребра (0,2), базуючись на досліджуваному графі E з однаковим кутом $\theta = \pi/4$, тоді отримаємо наступний стан квантового вектора $|\psi_3\rangle$:

$$|\psi_3\rangle = U_{0 \rightarrow 2}(\theta) |\psi_2\rangle \approx \begin{pmatrix} 0.103553 \\ 0.603553 \\ 0.250000 \\ 0.250000 \\ 0.250000 \\ 0.250000 \\ 0.603553 \\ 0.103553 \end{pmatrix}. \quad (2.39)$$

Після застосування блоку $U_{1 \rightarrow 2}(\pi/4)$ для останнього ребра (1,2) графа E отримаємо фінальний вектор стану $|\psi_4\rangle$:

$$|\psi_4\rangle = U_{1 \rightarrow 2}(\theta) |\psi_3\rangle \approx \begin{pmatrix} 0 \\ 0.461940 \\ 0.461940 \\ 0.270598 \\ 0.270598 \\ 0.461940 \\ 0.461940 \\ 0 \end{pmatrix}. \quad (2.40)$$

Отже, після послідовного застосування ребрових операторів для всіх ребер трикутника амплітуди базисних станів $|000\rangle$ та $|111\rangle$ практично зануляються, тоді як для конфігурацій з неоднаковими бітами формується підсилення.

Вимірювання в обчислювальному базисі задає дискретний розподіл імовірностей

$$P(x) = |\langle x | \psi_4 \rangle|^2, x \in \{0,1\}^3, \quad (2.41)$$

який для наведеного прикладу має вигляд:

$$\begin{aligned} P(000) &\approx 0; \\ P(111) &\approx 0; \\ P(001) &\approx 0.21339; \\ P(010) &\approx 0.21339; \\ P(101) &\approx 0.21339; \\ P(110) &\approx 0.21339. \end{aligned} \quad (2.42)$$

Інтерпретація отриманого бітового рядка $x = (x_0, x_1, x_2)$ як розбиття вершин на дві підмножини здійснюється за правилом: вершина v належить першій частині, якщо $x_v = 0$, та другій частині, якщо $x_v = 1$. Значення цільової функції Мах-Сут для незваженого графа визначається як $C(x)$ по (2.5).

Для трикутника максимум становить $C_{max} = 2$ та досягається для будь-якої не сталої конфігурації (тобто для всіх $x \notin 000, 111$). З отриманого розподілу випливає, що вимірювання з найбільшою імовірністю повертає рядки 001, 010, 101, 110, кожен з яких відповідає розрізу з двома перерізними ребрами, тобто забезпечує оптимальне значення $C(x) = 2$. Таким чином, квантова еволюція, сформована композицією блоків $CX - R_y - CX$ по ребрах графа, перерозподіляє вагу ймовірностей на користь оптимальних (максимальних) розрізів, що дозволяє на етапі класичного аналізу обрати розв'язок Мах-Сут без додаткових ітерацій оптимізації параметрів у межах даного демонстраційного прикладу.

Після виконання квантової схеми квантовий процесор повертає вибірку бітових рядків

$$b \in \{0,1\}^n, \quad (2.43)$$

разом з емпіричними частотами їх спостереження, які інтерпретуються як оцінки імовірностей відповідних конфігурацій. Сукупність отриманих результатів зручно представляти у вигляді гістограми вимірювань, що відображає імовірнісний вихід квантового обчислення.

Кожен бітовий рядок $b = (b_1, \dots, b_n)$ задає двомножинне розбиття множини вершин графа $G = (V, E, W)$ на підмножини

$$V_1(b) = \{v_i \in V \mid b_i = 1\}, V_2(b) = \{v_i \in V \mid b_i = 0\}, \quad (2.44)$$

які задовольняють умови $V_1 \cap V_2 = \emptyset$ та $V_1 \cup V_2 = V$. Таке подання безпосередньо відповідає формулюванню задачі максимального розрізу, в якій кожна вершина належить рівно одній з двох частин. Для оцінювання якості розрізу, що відповідає рядку b , обчислюється значення цільової функції

$$C(b) = \sum_{(i,j) \in E} w_{ij} \mathbf{1}[b_i \neq b_j], \quad (2.45)$$

де $\mathbf{1}[\cdot]$ – індикаторна функція, що дорівнює 1, якщо вершини i та j належать різним підмножинам, і 0 – в іншому випадку. Ця форма є концептуально прозорою і безпосередньо відображає кількість (або сумарну вагу) ребер, що перетинають розріз. Варто зауважити, що функція $C(b)$ є інваріантною відносно бітової інверсії: для комплементарного рядка $\bar{b} = (1 - b_1, \dots, 1 - b_n)$ виконується рівність $C(\bar{b}) = C(b)$. Отже, кожному розрізу відповідає щонайменше пара еквівалентних бітових рядків, що відрізняються лише перестановкою підмножин V_1 та V_2 . Крім того, для практичних графів часто спостерігається виродження оптимуму, коли декілька різних бітових конфігурацій забезпечують однакове максимальне значення цільової функції. Усі такі конфігурації є допустимими розв'язками задачі Max-Cut.

Класичний аналіз результатів квантового обчислення здійснюється у два послідовні етапи. На першому етапі з гістограми вимірювань відбираються бітові рядки з найбільшими емпіричними частотами, оскільки саме вони

відповідають конфігураціям, амплітуди яких були інтерференційно підсилені в процесі квантової еволюції. Обчислення цільової функції для низькочастотних результатів не є доцільним, оскільки вони з високою ймовірністю відповідають неоптимальним або близьким до шумових конфігураціям.

На другому етапі для відібраних кандидатів обчислюється точне значення $C(b)$, після чого обирається конфігурація (або множина конфігурацій) з максимальним значенням цільової функції. Таким чином, квантова частина алгоритму виконує імовірнісне підсилення перспективних розрізів, тоді як класичний етап забезпечує детермінований та однозначний вибір оптимального результату. Для ілюстрації розглянемо приклад графа E при умові, що квантовий процесор повернув найкращий результат $b = (0,0,1)$. Тоді бітовий вектор b задає двомножинне розбиття вершин графа

$$V_1(b) = v_2, V_2(b) = \{v_0, v_1\}. \quad (2.46)$$

У наведеному прикладі ребра, що перетинають розріз (V_1, V_2) , є $(0,2)$ та $(1,2)$, тоді як ребро $(0,1)$ є внутрішньомножинним. Отже, $C(b) = 2$, що відповідає оптимальному значенню задачі Max-Cut для трикутника, оскільки максимальний розріз у повному графі з трьох вершин дорівнює 2 та сходиться з імовірнісним розподілом математичної моделі. Таким чином, квантовий процесор генерує імовірнісний сигнал у вигляді розподілу бітових рядків, де оптимальні розв'язки мають суттєво вищі емпіричні частоти, а класичний аналіз перетворює цей сигнал на детерміноване рішення задачі Max-Cut шляхом відображення рядка b у розбиття (V_1, V_2) та точного обчислення вартості $C(b)$, тобто підсумку послідовність «квантове інтерференційне підсилення – частотний добір – детермінований підрахунок» утворює коректний, відтворюваний та обчислювально ефективний механізм отримання розв'язку задачі максимального розрізу.

Таким чином, у межах розглянутої математичної моделі квантово-гібридного алгоритму для задачі максимального розрізу було послідовно побудовано квантову еволюцію стану обчислювального регістру. Загальна

послідовність виконання запропонованого квантово-гібридного алгоритму, включно з етапами класичної підготовки даних, побудови квантової схеми, виконання квантових обчислень та постобробки результатів, схематично подана на рис. 2.20. Наведена схема узагальнює математичний опис, поданий у цьому розділі, та наочно ілюструє взаємодію між класичними й квантовими компонентами алгоритму.

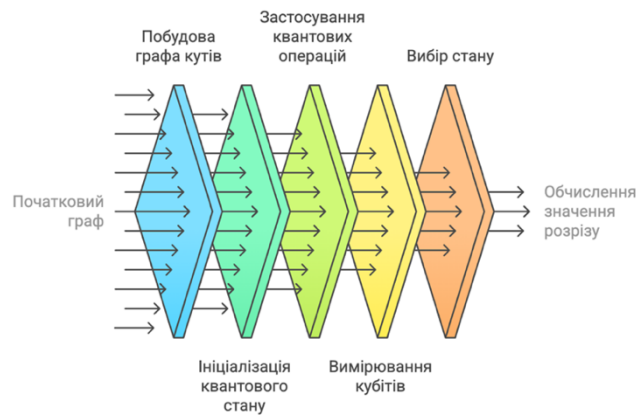


Рис. 2.20. Модель роботи алгоритму QASPA [5]

Початковий стан $|0\rangle^{\otimes n}$ переводиться у рівномірну суперпозицію за допомогою шару вентилів Адамара, що формує повний простір кандидатних бітових конфігурацій без упереджень щодо структури задачі. Подальша квантова еволюція визначається композицією унітарних операторів, асоційованих з ребрами графа, кожен з яких реалізується у вигляді блоку $CX - R_y - CX$ з фіксованим параметром повороту. Отриманий аналітичний опис дозволяє явно простежити, яким чином інформація про топологію графа та цільову функцію Max-Cut вбудовується у квантову схему через керовані однокубітні обертання. Показано, що застосування такого оператора для кожного ребра призводить до перерозподілу амплітуд у просторі базисних станів: конфігурації, що відповідають більшим значенням функції розрізу, підсилюються інтерференційно, тоді як конфігурації з меншою кількістю перерізаних ребер пригнічуються. Цей ефект підтверджується як аналітичним виведенням матричних операторів, так і чисельними результатами для тестового графа.

На завершальному етапі квантової частини алгоритму здійснюється вимірювання обчислювального регістру в обчислювальному базисі, внаслідок чого формується дискретний розподіл імовірностей на множині бітових рядків. Подальший класичний аналіз полягає у відборі найбільш імовірних конфігурацій та обчисленні для них значення цільової функції Max-Cut. За рахунок інтерференційного підсилення оптимальні або квазіоптимальні розрізи з'являються серед результатів вимірювання з істотно вищою ймовірністю, що дозволяє отримати коректний розв'язок без багатоітераційної оптимізації параметрів.

Висновки до розділу 2

У другому розділі розроблено квантово-гібридний алгоритм QASPA для розв'язання задачі максимального розрізу, який інтегрує класичну підготовку параметрів та квантову еволюцію стану з подальшою класичною обробкою результатів. На відміну від існуючих підходів, зокрема алгоритму QAOA, запропонований алгоритм базується на одноітераційній схемі виконання, що виключає необхідність багаторазової взаємодії між класичною та квантовою підсистемами.

У межах розробленого алгоритму запропоновано метод попереднього визначення параметрів квантової схеми на основі структурних характеристик графа, зокрема матриці суміжності та ваг ребер, що дозволяє формувати квантову схему без подальшої ітераційної оптимізації параметрів.

Розроблено метод перетворення графа задачі Max-Cut у форму, придатну для квантової обробки, який включає представлення графа у вигляді симетричної матриці суміжності, її нормування для зважених графів, а також генерацію матриці параметрів кутів ϕ . Встановлено, що використання фіксованого значення $\phi_{ij} = \pi/4$ для ребер графа забезпечує формування розподілу ймовірностей, орієнтованого на конфігурації з максимальним значенням розрізу при збереженні обмеженої глибини квантової схеми.

Запропоновано підхід до визначення кількості квантових запусків (shots) на основі афінно-експоненційної апроксимації, що дозволяє забезпечити стабільність отримання результатів в умовах шуму та обмежених ресурсів квантових пристроїв.

Розроблено метод формування квантової схеми, який відображає структуру графа у параметризовану унітарну еволюцію, що включає ініціалізацію стану за допомогою шару гейтів Адамара та подальше застосування операцій типу $CX-R_y-CX$, які кодують зв'язки між вершинами графа та впливають на розподіл імовірностей вимірювань.

Таким чином, запропонований алгоритм QASPA та відповідні методи його реалізації формують узгоджений підхід до розв'язання задачі Max-Cut, орієнтований на зменшення кількості квантово-класичних ітерацій та адаптацію до обмежень сучасних квантових обчислювальних пристроїв.

Матеріали розділу 2 подано в таких публікації автора [5].

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ QASPA

3.1. Програмна реалізація QASPA

На сучасному етапі розвитку квантових технологій, який зазвичай окреслюється терміном NISQ, вибір програмної платформи для реалізації квантових алгоритмів набуває особливого значення. Цей етап характеризується обмеженою кількістю доступних кубітів, наявністю шуму та апаратними похибками, що зумовлює необхідність адаптації алгоритмічних підходів і використання квантово-гібридних обчислювальних парадигм. У зв'язку з цим сформувався спектр програмних платформ (рис. 3.1), кожна з яких пропонує власні інструменти та підходи до розробки квантових алгоритмів. Аналіз таких платформ є важливим для здійснення обґрунтованого вибору середовища реалізації, що відповідає специфіці досліджуваної задачі, зокрема розробці та експериментальній валідації алгоритму QASPA.

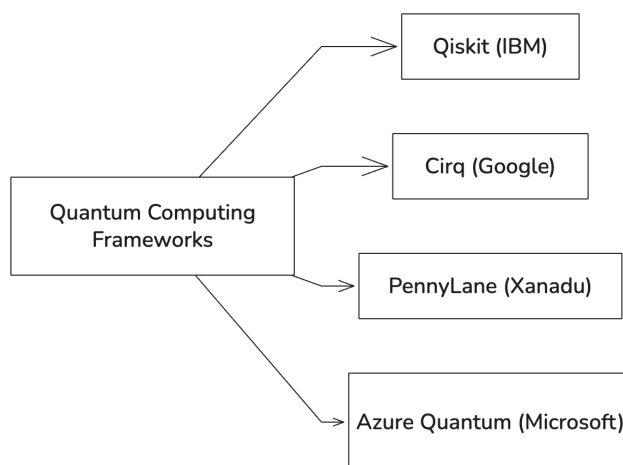


Рис. 3.1. Платформи для квантового програмування

Серед провідних програмних фреймворків у галузі квантових обчислень доцільно виділити Qiskit, Cirq, PennyLane та Azure Quantum, кожен із яких характеризується власною концептуальною спрямованістю та набором інструментів. Зокрема, Qiskit як відкритий програмний стек надає засоби для

побудови, симуляції та виконання квантових схем, а також забезпечує доступ до реальних квантових процесорів компанії IBM [104].

Фреймворк Cirq орієнтований на більш детальний контроль квантових операцій, що дозволяє здійснювати дослідження з урахуванням особливостей апаратної реалізації та архітектури квантових пристроїв [19]. PennyLane, у свою чергу, інтегрується з бібліотеками машинного навчання, що забезпечує можливість реалізації варіаційних квантових алгоритмів і гібридних моделей оптимізації [93]. Azure Quantum є хмарною платформою, яка надає уніфікований доступ до різних квантових апаратних і програмних рішень від кількох постачальників, формуючи єдине середовище для виконання квантових обчислень [42]. Розвиток зазначених платформ сприяє переходу від теоретичних досліджень до практичної реалізації квантових алгоритмів.

Вибір платформи Qiskit для програмної реалізації алгоритму QASPA та проведення експериментальних досліджень зумовлений поєднанням функціональних можливостей та відповідністю вимогам сучасних квантових обчислювальних систем. Qiskit являє собою комплексну екосистему, що охоплює основні етапи розробки квантових алгоритмів – від побудови квантових схем до їх виконання на симуляторах і реальному квантовому апаратному забезпеченні.

Архітектура Qiskit базується на модульному підході та включає взаємопов'язані компоненти (рис. 3.2), зокрема Qiskit Terra для побудови та трансформації квантових схем, Qiskit Aer для симуляції квантових обчислень із можливістю врахування шумових процесів, а також інші інструменти, орієнтовані на реалізацію прикладних алгоритмів і аналіз характеристик квантових пристроїв. Така організація забезпечує можливість керування складністю квантово-гібридного алгоритму QASPA та адаптації його реалізації до різних умов експериментального дослідження. Зокрема, для реалізації алгоритму QASPA, що передбачає побудову параметризованих квантових схем із керованими обертаннями та подальшу класичну обробку

результатів, інструментарій Qiskit Terra забезпечує необхідні засоби для конструювання та модифікації квантових схем.

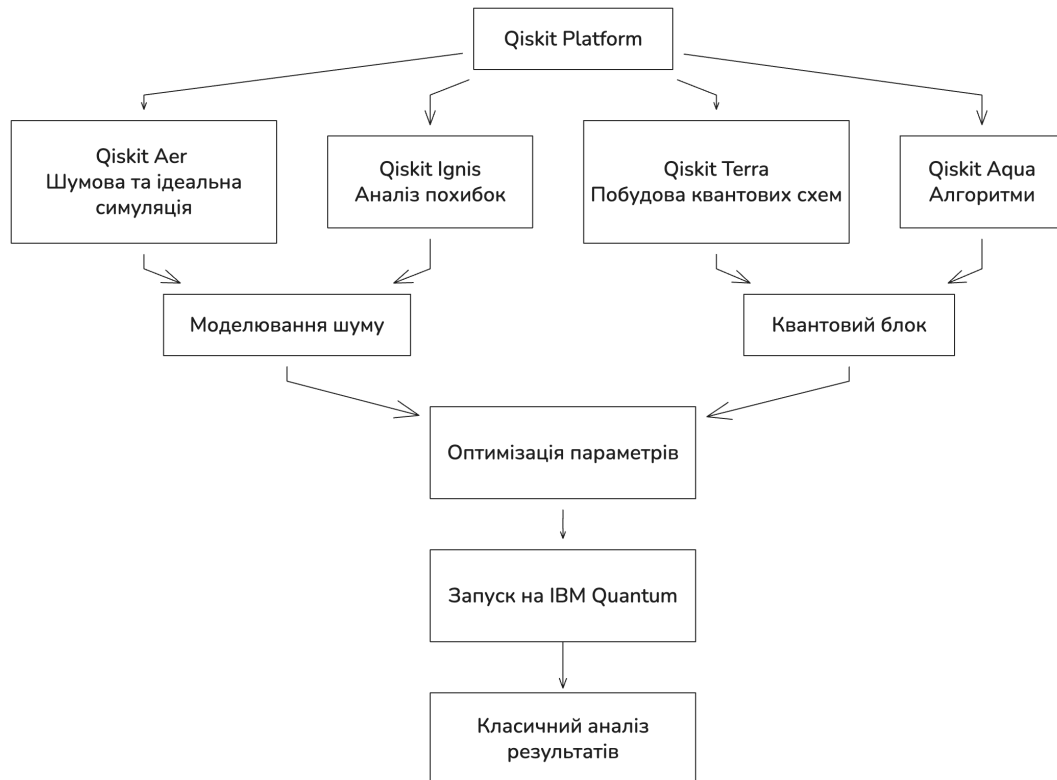


Рис. 3.2. Архітектура Qiskit

Можливості симуляції, які надає Qiskit Aer, дозволяють здійснювати як ідеалізовану безшумну симуляцію, так і моделювання впливу шуму, що є важливим в умовах NISQ-ери [89]. Використання таких симуляторів, як Qasm simulator, дає змогу виконувати серії експериментів та отримувати статистичні дані в контрольованому середовищі. Це, у свою чергу, забезпечує можливість аналізу стійкості алгоритму QASPA до шумових впливів, оцінки його масштабованості та попереднього налаштування параметрів перед виконанням на реальному квантовому апаратному забезпеченні.

Важливою характеристикою платформи Qiskit є інтегрований доступ до реальних квантових процесорів через сервіс IBM Quantum Experience [27]. Незважаючи на обмеження сучасних квантових пристроїв, пов'язані з кількістю кубітів та впливом декогеренції [38], можливість виконання алгоритму QASPA на реальному квантовому обладнанні забезпечує перевірку отриманих результатів в умовах фізичних обмежень. Отримані емпіричні дані

дозволяють здійснювати оцінку ефективності алгоритму та аналіз його застосовності до задач комбінаторної оптимізації, що є важливим для підтвердження результатів дослідження [60]. Водночас наявність розвиненої спільноти користувачів Qiskit, а також доступної документації та освітніх ресурсів сприяє засвоєнню інструментарію та впровадженню сучасних підходів квантових обчислень у дослідницьку практику.

Таким чином, з урахуванням відкритої архітектури, функціональних можливостей, засобів симуляції та доступу до реального квантового апаратного забезпечення, платформа Qiskit є обґрунтованим вибором для програмної реалізації та експериментального дослідження алгоритму QASPA, забезпечуючи умови для отримання відтворюваних наукових результатів у межах даної дисертаційної роботи.

Програмна реалізація квантово-гібридного алгоритму QASPA розглядається як розвиток теоретичних положень, сформульованих у розділі 2, та спрямована на перетворення запропонованих підходів у функціональний обчислювальний інструмент. Така реалізація відповідає гібридній парадигмі, у межах якої класичні обчислювальні ресурси використовуються для підготовки задачі та обробки результатів, тоді як квантовий процесор (симулятор або реальний пристрій) виконує квантові операції. Наявність програмного прототипу дозволяє здійснювати емпіричну перевірку ефективності, масштабованості та стійкості алгоритму QASPA в умовах NISQ-ери. Для реалізації алгоритму також використовувалися бібліотеки NetworkX для роботи з графами, NumPy для чисельних обчислень та Matplotlib для візуалізації результатів.

Центральний програмний модуль `qaspa_maxcut` реалізує побудову квантової схеми, тоді як функції `build_angle_graph`, `normalize_angle` та `compute_cut_value` забезпечують виконання відповідних класичних перетворень. Управління процесом виконання, як у локальному середовищі, так і з використанням віддалених обчислювальних ресурсів, здійснюється функціями `run_qaspa_maxcut_local` та `run_qaspa_maxcut_cloud`, які охоплюють

основні етапи обчислювального процесу – від формування графового подання задачі до отримання та інтерпретації результатів.

Першим етапом програмної реалізації є класична підготовка вхідних даних. Вхідний граф, представлений у форматі NetworkX Graph, перетворюється у матрицю суміжності типу NumPy ndarray за допомогою функції `nx.to_numpy_array`. Такий підхід дозволяє ефективно виконувати чисельні операції над вагами ребер та аналіз зв'язків між вершинами.

Ключовим елементом підготовки є генерація матриці кутів, яка безпосередньо параметризує квантову схему. Функція `build_angle_graph` створює цю матрицю, застосовуючи функцію `normalize_angle` до кожного елемента матриці суміжності. Відповідно до підрозділу 2.2, нормування та вибір фіксованого кута $\phi_{ij} = \pi/4$ для всіх наявних ребер є критично важливим. У коді функція `normalize_angle` реалізована таким чином, що незалежно від вхідного значення (яке представляє вагу ребра), вона завжди повертає $\pi/4$:

```
def normalize_angle(value):
    return np.pi / 4
```

Такий підхід, як зазначено у теоретичній моделі, забезпечує помітний амплітудний зсув на заданий кут, сприяючи інтерференційному підсиленню конфігурацій, що відповідають максимізації розрізу. Це також дозволяє уникнути надмірної глибини квантової схеми та узгоджується з апаратною транспіляцією на NISQ-пристроях. Хоча в теоретичному розділі згадується можливість вагової модуляції ($\phi_{ij} = \frac{\pi}{4} \cdot \widetilde{w}_{ij}$), поточна реалізація зосереджується на бінарному випадку Max-Cut, де всі ваги ребер вважаються одиничними, що робить фіксований кут $\pi/4$ доцільним.

Побудова квантової схеми є ключовим етапом алгоритму QASPA і реалізована у функції `qaspa_maxcut`; цей процес відображає моделі та методи, описані в підрозділі 2.3. Квантова схема формується відповідно до кількості вершин графа, при цьому кожній вершині відповідає один логічний кубіт, що

забезпечує пряме відображення структури графа на топологію квантової схеми та не потребує введення додаткових допоміжних кубітів.

```
def qaspa_maxcut(graph):
    num_nodes = len(graph)
    angle_graph = build_angle_graph(graph)
    qc = QuantumCircuit(num_nodes, num_nodes)

    # Ініціалізація глобальної суперпозиції
    for i in range(num_nodes):
        qc.h(i)

    # Кодування ребер графа через керовані обертання
    for i in range(num_nodes):
        for j in range(i, num_nodes):
            if graph[i][j] > 0: # Якщо існує ребро
                angle = angle_graph[i][j]
                qc.cx(i, j)
                qc.ry(angle, j)
                qc.cx(i, j)
                qc.barrier()

    qc.barrier()
    qc.measure_all()

    return qc
```

Алгоритм починається з ініціалізації всіх кубітів у стані рівномірної суперпозиції. Це досягається шляхом застосування вентиля Адамара ($qc.h(i)$) до кожного кубіта. Як пояснюється в підрозділі 2.3, вентиль Адамара переводить стан кожного кубіта з північного полюса сфери Блоха на екватор, створюючи рівні амплітуди для станів $|0\rangle$ та $|1\rangle$. Це є передумовою для квантового паралелізму, забезпечуючи рівноймовірне представлення всіх бітових конфігурацій перед внесенням структури графа.

Наступним кроком є кодування ребер графа в унітарну динаміку, що здійснюється через керовані амплітудні обертання навколо осі O_y . Для кожної пари індексів (i, j) з верхнього трикутника матриці суміжності, де існує ребро ($graph[i][j] > 0$), застосовується спеціальний тригейтовий блок: $qc.cx(i, j)$, $qc.ry(angle, j)$, $qc.cx(i, j)$. Цей блок є апаратно-сумісною реалізацією керованого обертання ($CRY_{i \rightarrow j}(\phi_{ij})$), де кут обертання $angle$ береться з раніше

згенерованої `angle_graph`. Вибір гейта R_y мотивується його фізичною дією: обертання навколо осі O_y змінює амплітуди, безпосередньо впливаючи на розподіл ймовірностей вимірювань та підштовхуючи пари бітів до антикорельованих значень, що є бажаним для задачі максимального розрізу. Використання контрольованих гейтів (CNOT, `qc.cx`) у цьому триплеті створює заплутаність та умовно застосовує обертання, забезпечуючи, що операція на таргет-кубіті відбувається лише тоді, коли контрольний кубіт знаходиться у стані $|1\rangle$, і замикаючи керування, відновлюючи базис контролю.

Після застосування всіх необхідних обертань, в кінці схеми додається бар'єр (`qc.barrier()`) для візуального розділення етапів, а потім виконується вимірювання всіх кубітів (`qc.measure_all()`). Це переводить квантовий стан у класичний бітовий рядок, який потім аналізується.

Програмний прототип QASPA передбачає два основні режими виконання: локальну симуляцію та виконання на хмарних квантових ресурсах. Ці режими реалізовані функціями `run_qaspa_maxcut_local` та `run_qaspa_maxcut_cloud` відповідно.

```
def run_qaspa_maxcut_local(G, debug=False):
    graph = nx.to_numpy_array(G, dtype=int)
    n = len(graph)
    shots = int(np.ceil(0.03047 * (2 ** n) + 1016.2))
    qc = qaspa_maxcut(graph)
    simulator = AerSimulator()
    compiled = transpile(qc, simulator)
    result = simulator.run(compiled, shots=shots).result()
    counts = result.get_counts()
    sorted_counts = sorted(counts.items(), key=lambda item: item[1], reverse=True)
    top_4_results = []
    for i, (bitstring, count) in enumerate(sorted_counts[:4]):
        solution = bitstring.split(' ')[0][::-1]
        cut_value = compute_cut_value(solution, graph)
        top_4_results.append({
            'rank': i + 1,
            'cut_value': cut_value,
            'solution': solution,
            'count': count
        })
    top_4_results.sort(key=lambda x: x['cut_value'], reverse=True)
```

```

best_cut = top_4_results[0]['cut_value']
best_solution = top_4_results[0]['solution']
if debug:
    print("Final circuit:\n", qc)
    print("Best solution:", best_solution[::-1])
    print("Best cut value:", best_cut)
    colors = ['r' if best_solution[node] == '0' else 'b' for node in G]
    nx.draw(G, with_labels=True, node_color=colors)
    plt.savefig("qaspa_maxcut.png")
    res = {k.split(' ')[0]: v for k, v in counts.items()}
    plot_histogram(res, figsize=(12, 8)).savefig("qaspa_maxcut_counts.png")
return best_cut, best_solution

```

Для локальної симуляції (`run_qaspa_maxcut_local`) квантова схема, побудована функцією `qaspa_maxcut`, передається до симулятора `AerSimulator`. Перед виконанням схема піддається транспіляції (`transpile(qc, simulator)`), що забезпечує її адаптацію до особливостей цільового середовища виконання. Така процедура дозволяє враховувати обмеження симулятора або квантового пристрою та оптимізувати структуру схеми.

Кількість запусків (`shots`) є параметром, що визначає кількість повторних виконань квантової схеми з метою збору статистичних результатів вимірювань. Відповідно до підходу, описаного у підрозділі 2.2, значення `shots` визначається з урахуванням необхідності виявлення домінуючих конфігурацій у розподілі результатів, особливо зі зростанням розміру графа. У програмній реалізації значення `shots` обчислюється відповідно до заданої моделі. Результатом виконання є структура даних `counts`, яка містить бітові рядки та відповідні частоти їх появи.

Класична постобробка результатів реалізована у функції `compute_cut_value` та у логіці функцій виконання. Після отримання структури даних `counts`, що відображає розподіл результатів вимірювань, здійснюється відбір підмножини бітових рядків із найбільшими частотами появи. Зокрема, формується множина з чотирьох конфігурацій, які відповідають максимальним значенням частот у розподілі. Серед відібраних конфігурацій визначається розв'язок, що відповідає максимальному значенню функції

розрізу, який обчислюється за допомогою функції `compute_cut_value` (наприклад, `best_cut = top_4_results[0]['cut_value']`). Такий підхід базується на припущенні, що розподіл результатів вимірювань містить підвищені ймовірності для конфігурацій, близьких до оптимального розв'язку. Інтерпретація отриманого бітового рядка як двомножинного розбиття вершин графа V_1, V_2 виконується відповідно до підходу, описаного у підрозділі 2.3.

Функція `compute_cut_value` обчислює вартість розрізу для заданого бітового рядка та графа:

```
def compute_cut_value(bitstring, graph):
    cut_value = 0
    n = len(graph)
    for i in range(n):
        for j in range(i + 1, n):
            if graph[i][j] > 0: # Якщо є ребро
                if bitstring[i] != bitstring[j]: # Якщо вершини в різних підмножинах
                    cut_value += graph[i][j]
    return cut_value
```

Дана функція виконує перевірку кожного ребра графа: якщо вершини, які воно з'єднує, належать до різних підмножин (що визначається за значеннями бітів у `bitstring`), вага відповідного ребра додається до значення `cut_value`. Така процедура відповідає математичній формалізації обчислення вартості розрізу, наведеній у підрозділі 2.3. Додатково класична постобробка враховує комплементарну симетрію розв'язків та можливу множинність оптимальних конфігурацій, що є характерною властивістю задачі Max-Cut.

Завершальні етапи програмної реалізації включають візуалізацію результатів (зокрема при активації параметра `debug=True`), у межах якої граф відображається відповідно до отриманого розбиття вершин, а також формується гістограма результатів вимірювань для подальшого аналізу. Такі допоміжні засоби сприяють інтерпретації поведінки алгоритму та перевірці отриманих результатів.

Таким чином, програмна реалізація алгоритму QASPA, із використанням можливостей Qiskit та інших Python-бібліотек, відображає

квантово-гібридний підхід, запропонований у дисертаційному дослідженні. Вона охоплює етапи підготовки графових даних, побудови та виконання квантових схем, а також подальшої обробки результатів, що забезпечує можливість проведення експериментальних досліджень і перевірки властивостей алгоритму. Запропонований програмний прототип використовується для аналізу ефективності та масштабованості алгоритму QASPA, що розглядається у наступних підрозділах цього розділу.

3.2. Запуск алгоритму на квантовому симуляторі та на реальному квантовому пристрої

Для проведення симуляційного експерименту було обрано неорієнтований незважений граф із 5 вершинами та 8 ребрами, структура якого відповідає циклу з чотирьох вершин, додатково з'єднаних із центральною п'ятою вершиною. Граф визначається множиною ребер $[(0,1),(1,2),(2,3),(3,0),(0,4),(1,4),(2,4),(3,4)]$. Обраний граф характеризується регулярною структурою зв'язків і наявністю симетричних підструктур, що дозволяє дослідити поведінку алгоритму QASPA на прикладі невеликого, але структурно нетривіального випадку. Класичне представлення графа у форматі NetworkX Graph використовується як вхідні дані для подальших етапів обчислювального процесу.

Відповідно до підрозділу 2.3, першим етапом є класична підготовка графа та генерація матриці кутів. У даному випадку, оскільки граф незважений, функція `normalize_angle` присвоює фіксоване значення $\pi/4$ для кожного існуючого ребра, що є критично важливим для унітарної параметризації квантової схеми. Наступним кроком є побудова квантової схеми за допомогою функції `qaspa_maxcut`, для 5 вершин графа створюється квантовий регістр з 5 кубітів, далі ініціалізується схема застосуванням вентилів Адамара до кожного кубіта (`qc.h(i)`), що переводить систему у стан рівномірної суперпозиції всіх можливих $2^5 = 32$ класичних конфігурацій. Цей крок є фундаментальним для забезпечення квантового паралелізму, надаючи

рівномовірне представлення всіх потенційних розв’язків задачі Max-Cut до внесення специфічної структури графа. Після ініціалізації суперпозиції, алгоритм кодує ребра графа в унітарну динаміку, застосовуючи до кожного ребра спеціалізований тригейтовий блок $qc.cx(i, j)$, $qc.rz(\text{angle}, j)$, $qc.cx(i, j)$. Використання контрольованих X-гейтів ($qc.cx$) створює запутаність між кубітами, що відповідають зв’язаним вершинам графа, і забезпечує умовне застосування R_y обертання, ефективно відображаючи топологію графа у квантову схему. Бар’єри ($qc.barrier()$) додаються для логічного та візуального структурування схеми.

Після виконання всіх операцій, всі кубіти вимірюються ($qc.measure_all()$), що призводить до колапсу квантового стану до класичного бітового рядка. Загальна структура отриманої квантової схеми, що включає ініціалізацію, застосування обертань для кодування ребер та вимірювання, представлена на рисунку 3.3.

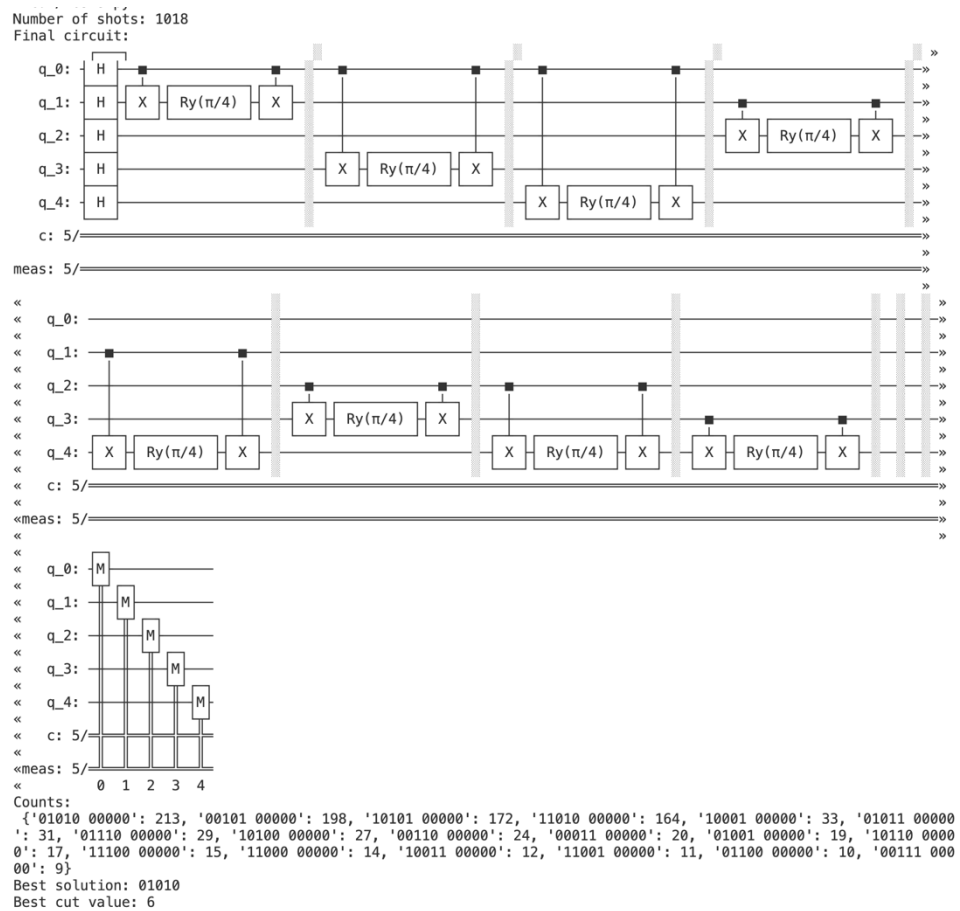


Рис. 3.3. Квантова схема для графа з п'ятьма вершинами

Виконання квантової схеми здійснюється локально з використанням симулятора AerSimulator у межах функції `run_qaspa_maxcut_local`. Кількість запусків (shots) для симуляції встановлено на рівні 1018. Обране значення відповідає моделі масштабування, представлений у підрозділі 2.2, та забезпечує достатню кількість вимірювань для виокремлення домінуючих конфігурацій у розподілі результатів.

Результатом симуляції є структура даних `counts`, що містить бітові рядки та частоти їх появи за результатами 1018 вимірювань. Бітовий рядок із максимальною частотою розглядається як кандидат на оптимальний розв'язок. У даному експерименті такою конфігурацією є бітовий рядок 01010 (з частотою 213), а відповідне значення функції розрізу, обчислене за допомогою функції `compute_cut_value`, становить 6.

Отримано наступні результати:

Counts:

```
{'01010 00000': 213, '00101 00000': 198, '10101 00000': 172, '11010 00000': 164, '10001 00000': 33, '01011 00000': 31, '01110 00000': 29, '10100 00000': 27, '00110 00000': 24, '00011 00000': 20, '01001 00000': 19, '10110 00000': 17, '11100 00000': 15, '11000 00000': 14, '10011 00000': 12, '11001 00000': 11, '01100 00000': 10, '00111 00000': 9}
```

Best solution: 01010

Best cut value: 6

Гістограма розподілу результатів вимірювань (рис. 3.4) наочно демонструє піки, що відповідають найбільш імовірним бітовим рядкам. Згідно з математичною моделлю описаною у підрозділі 2.2, бітовий вектор 01010 задає розбиття вершин графа на дві підмножини, 0 інтерпретується як належність до першої підмножини V_0 , а 1 – до другої підмножини V_1 , то для 01010 отримуємо:

- V_0 = вершина 0, вершина 2, вершина 4, (де біт = 0);
- V_1 = вершина 1, вершина 3, (де біт = 1).

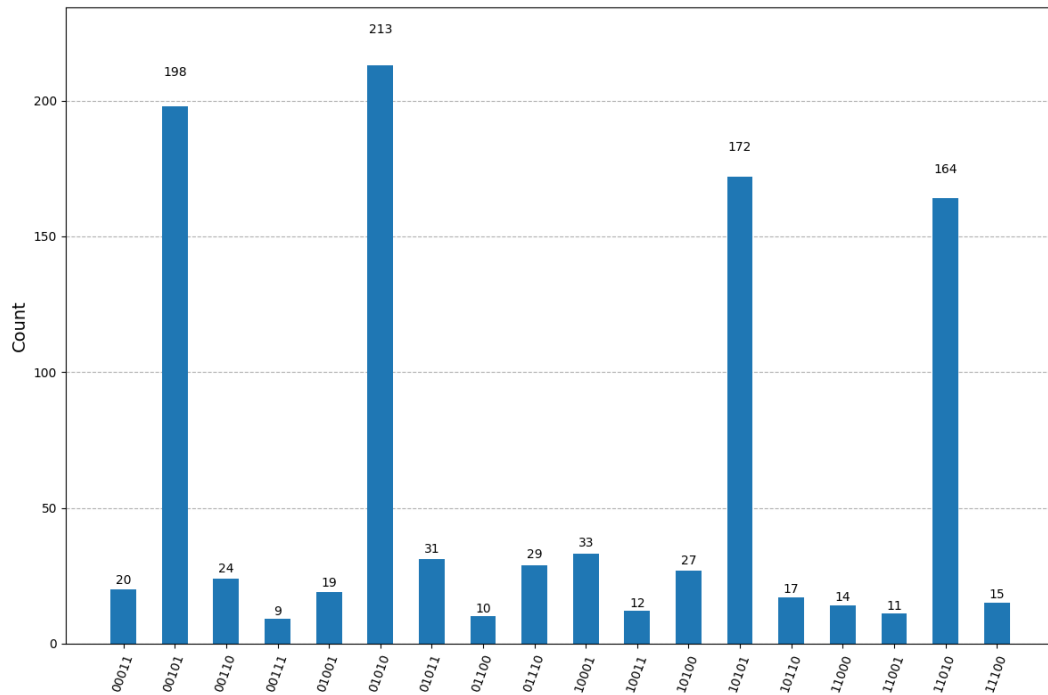


Рис. 3.4. Гістограма розподілу результатів вимірювань для алгоритму QASPA

Розглянемо детально, які саме ребра графа перетинають розріз, визначений бітовим вектором 01010:

- Вершина 0 (належить до V_0), вершина 1 (належить до V_1) – ребро перетинає розріз;
- Вершина 1 (V_1), вершина 2 (V_0) – ребро перетинає розріз;
- Вершина 2 (V_0), вершина 3 (V_1) – ребро перетинає розріз;
- Вершина 3 (V_1), вершина 0 (V_0) – ребро перетинає розріз;
- Вершина 0 (V_0), вершина 4 (V_0) – ребро не перетинає розріз;
- Вершина 1 (V_1), вершина 4 (V_0) – ребро перетинає розріз;
- Вершина 2 (V_0), вершина 4 (V_0) – ребро не перетинає розріз;
- Вершина 3 (V_1), вершина 4 (V_0) – ребро перетинає розріз.

Таким чином, до розрізу входять ребра між вершинами 0-1, 1-2, 2-3, 3-0, 1-4 та 3-4. Решта ребер (0-4, 2-4) залишаються в межах однієї підмножини та не враховуються у вартості розрізу. Загальна кількість ребер, що перетинають розріз, становить 6 (0-1, 1-2, 2-3, 3-0, 1-4, 3-4), що повністю збігається зі значенням Best cut value: 6, отриманим програмно.

Важливо відзначити, що серед домінуючих розв'язків у counts присутні також бітові рядки 10101 та 11010 з високими частотами, 10101 є комплементарним до 01010, що, згідно з підрозділом 2.3, має призводити до однакової вартості розрізу завдяки комплементарній симетрії задачі Max-Cut. Відповідно, для 10101 розбиття буде V'_0 = вершина 1, вершина 3 та V'_1 = вершина 0, вершина 2, вершина 4, що дасть той самий розріз з вартістю 6, це підтверджує теоретичні передбачення алгоритму.

Візуалізація графа з розфарбованими вершинами відповідно до знайденого розбиття (01010) представлена на рис. 3.5, де червоні та сині вузли позначають належність до різних підмножин розрізу.

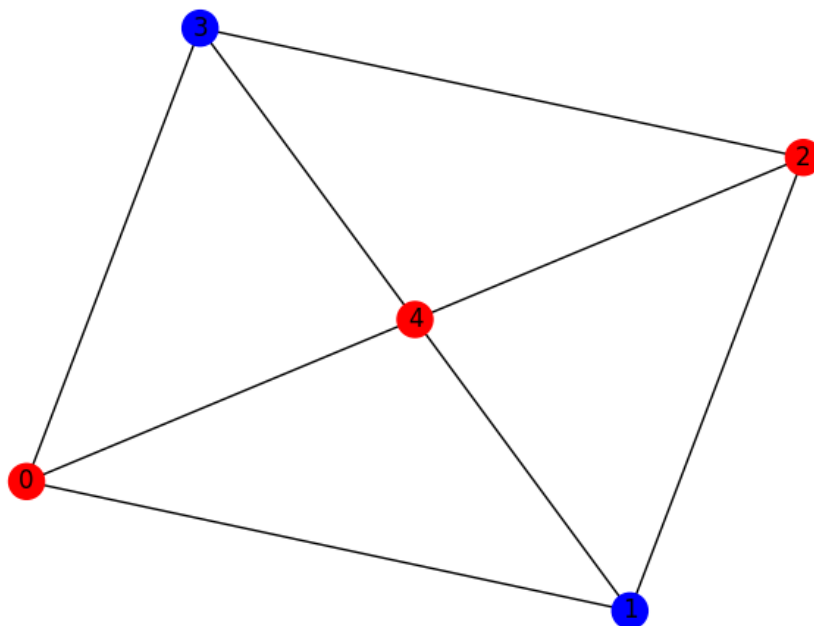


Рис. 3.5. Тестовий граф із візуалізацією знайденого розрізу 01010

Аналіз гістограми результатів вимірювань показує, що у розподілі спостерігається підвищена частота бітових конфігурацій, які відповідають максимальному розрізу, а також їх комплементарних варіантів. Такі конфігурації з'являються значно частіше порівняно з іншими станами, що узгоджується з очікуваною поведінкою квантової схеми, пов'язаною з інтерференційними ефектами у процесі еволюції квантового стану.

Отримані результати узгоджуються з положеннями, викладеними у підрозділі 2.2, де розглядається здатність параметризованої квантової схеми змінювати розподіл ймовірностей вимірювань на користь конфігурацій, що відповідають високим значенням цільової функції. Навіть для відносно невеликої кількості кубітів у розподілі чітко спостерігаються домінуючі піки, що вказує на структурованість отриманих результатів.

Таким чином, результати симуляційного експерименту для тестового графа з 5 вершинами демонструють працездатність програмної реалізації алгоритму QASPA та її узгодженість із теоретичними моделями, представленими у розділі 2. Отримані дані свідчать про можливість виявлення високоякісних розв'язків задачі Max-Cut із використанням запропонованого підходу та ілюструють характер формування розподілу ймовірностей у квантово-гібридному алгоритмі.

Перехід від симуляційних експериментів до виконання квантового алгоритму на реальному апаратному забезпеченні є важливим етапом оцінювання його практичної придатності, ефективності та стійкості в умовах сучасної NISQ-ери. Хоча квантові симулятори, зокрема Qiskit Aer, дозволяють проводити контрольовані дослідження та аналізувати поведінку алгоритму, вони не відображають повною мірою сукупність фізичних обмежень, притаманних реальним квантовим процесорам. До таких обмежень належать обмежена кількість доступних кубітів, вплив шуму, кінцеві часи когерентності, процеси декогеренції, а також обмеження топології зв'язності, що безпосередньо впливають на результати виконання квантових схем. У зв'язку з цим виконання алгоритму QASPA на фізичному квантовому обладнанні розглядається як необхідний етап експериментальної перевірки його властивостей та аналізу поведінки в умовах реального середовища виконання.

Обрання платформи IBM Quantum Platform як експериментальної бази для запуску алгоритму QASPA на реальному апаратному забезпеченні зумовлене її функціональними можливостями, розглянутими у підрозділі 3.1.

Платформа забезпечує хмарний доступ до квантових обчислювальних ресурсів, надаючи можливість віддаленого виконання квантових схем на різних фізичних процесорах. Взаємодія з обчислювальним середовищем реалізується за допомогою бібліотеки Qiskit Runtime (qiskit-ibm-runtime), яка надає інтерфейс для роботи з API платформи. Засоби керування дозволяють відстежувати статус виконання завдань, отримувати результати обчислень та здійснювати розподіл доступних ресурсів. Приклад інтерфейсу, що відображає історію виконання завдань, наведено на рис. 3.6, де представлено інформацію про статус виконання, використані квантові процесори та тривалість обчислень. Застосування даної інфраструктури забезпечує можливість проведення експериментальних досліджень алгоритму QASPA на різних апаратних конфігураціях та отримання емпіричних результатів для подальшого аналізу.

The screenshot displays the IBM Quantum Platform interface. On the left, there's a sidebar with 'API key' and 'Instances' sections. The main area shows 'My recent workloads' as a table with columns: ID, Status, Instance, Created, QPU, and Usage. Below this, a 'QPUs' section shows a list of available quantum processors with their respective qubit counts and error rates.

ID	Status	Instance	Created	QPU	Usage
64g2p8b9c73...	Completed	test	11/21/25, 11:11 AM	ibm_torino	2s
64g2p8b9c73...	Completed	test	11/21/25, 11:09 AM	ibm_mumbai	2s
64g2p8b9c73...	Completed	test	11/18/25, 9:35 AM	ibm_mumbai	6s
64g2p8b9c73...	Completed	test	11/18/25, 8:22 AM	ibm_mumbai	2s
64g2p8b9c73...	Completed	test	11/17/25, 11:18 PM	ibm_mumbai	2s
64g2p8b9c73...	Completed	test	11/17/25, 11:15 PM	ibm_torino	3s
62p8b9c73...	Cancelled	Pay-as-you-go	8/22/25, 12:18 AM	ibm_pittsburgh	0s

QPUs	View all
ibm_kingston 334q 1 error (2)	
ibm_pittsburgh 154q 1 error (3)	
ibm_tec 154q 1 error (2)	
ibm_mumbai 154q 1 error (2)	
ibm_torino 133q 1 error (1)	

Рис. 3.6. Вебінтерфейс IBM Quantum Platform

Квантові процесори IBM базуються на технології надпровідних трансмонних кубітів, що є одним із поширених підходів до фізичної реалізації квантових обчислень. Незважаючи на поступовий розвиток цієї технології, кожен квантовий процесор характеризується індивідуальними параметрами та обмеженнями, які впливають на виконання квантових алгоритмів. О ключових характеристик належать кількість доступних кубітів, топологія зв'язності (coupling map), часи когерентності (T_1 та T_2), вірність гейтів (gate fidelity) для одно- та двокубітних операцій, а також похибки зчитування (readout error). Урахування зазначених параметрів є необхідним для коректного

транспілювання квантових схем та їх подальшого виконання на фізичному обладнанні.

Для експериментальної валідації алгоритму QASPA були обрані квантові процесори сімейства IBM Heron, зокрема моделі Heron R1 та Heron R2. Зазначені процесори належать до сучасної архітектурної лінійки квантових пристроїв IBM та характеризуються розширеною зв'язністю і можливостями масштабування порівняно з попередніми поколіннями. Процесори сімейства Heron містять понад 100 кубітів і реалізують топологію типу heavy-hexagonal, яка забезпечує ефективну організацію зв'язків між кубітами. Така архітектура спрямована на зменшення необхідності використання додаткових операцій маршрутизації (зокрема SWAP-гейтів) під час транспіляції квантових схем, що є важливим для алгоритмів, які потребують значної кількості двокубітних взаємодій.

Характеристики процесорів, зокрема час життя збудженого стану (T_1), час фазової когерентності (T_2) та вірність квантових гейтів, впливають на тривалість збереження квантової інформації та точність виконання операцій, що, у свою чергу, відображається на якості отриманих результатів. З огляду на ці параметри, процесори Heron R1 та Heron R2 використовуються для експериментального дослідження алгоритму QASPA на графах середньої розмірності, які потребують реалізації багатокубітних взаємодій.

У ході дослідження було також здійснено спробу виконання алгоритму QASPA на квантовому процесорі IBM Eagle. Даний процесор, який містить 127 кубітів, представляє один із напрямів масштабування квантових обчислювальних систем і використовує архітектуру з гексагональною топологією зв'язності. Незважаючи на наявність значної кількості кубітів, виконання алгоритму QASPA на процесорі Eagle було ускладнене на етапі транспіляції квантової схеми. Основною причиною стало обмеження, пов'язане з відповідністю базового набору гейтів та топології зв'язності процесора вимогам, що виникають при реалізації параметризованих двокубітних операцій, характерних для алгоритму QASPA. Алгоритм QASPA

активно використовує тригейтовий блок CX-Ry-CX для кодування ребер графа. Цей триплет є апаратно-сумісною реалізацією керованого обертання $CRY_{i \rightarrow j}(\phi_{ij})$. Незважаючи на наявність у Qiskit засобів транспіляції для адаптації логічних схем до фізичних обмежень, у випадку процесора IBM Eagle виникли труднощі з ефективним відображенням цього гейт-триплету на базовий набір вентилів і топологію зв'язності без суттєвого ускладнення квантової схеми.

Однією з можливих причин є специфіка базового набору гейтів процесора Eagle, у межах якого реалізація двокубітних операцій може вимагати додаткового розкладу на елементарні операції, що призводить до збільшення глибини схеми. Крім того, топологія зв'язності, попри значну кількість кубітів, може бути неоптимальною для реалізації попарних взаємодій, необхідних для алгоритму QASPA, що зумовлює потребу у вставці додаткових операцій маршрутизації (зокрема SWAP-гейтів). У таких умовах транспіляція логічної схеми призводить до збільшення глибини квантового кола та накопичення шуму, що ускладнює отримання достовірних результатів. З огляду на це, ефективність реалізації алгоритму QASPA значною мірою залежить від відповідності між структурою задачі, топологією квантового процесора та доступним набором базових гейтів.

У межах проведених експериментів процесори сімейства IBM Heron (Heron R1 та Heron R2) виявилися більш придатними для реалізації алгоритму QASPA, що пов'язано з їх топологічними характеристиками та підтримкою двокубітних операцій, необхідних для кодування структури графа. Отримані результати узгоджуються з тим, що ефективність реалізації квантових алгоритмів у NISQ-умовах визначається не лише кількістю доступних кубітів, але й архітектурною відповідністю між апаратним забезпеченням і алгоритмічними вимогами.

Виконання алгоритму QASPA на реальному квантовому комп'ютері, хоча концептуально подібне до симуляції з точки зору багаторазового

виконання квантової схеми (shots) для збору статистики, потребує більш складного протоколу внаслідок шумової природи NISQ-пристроїв. Квантові стани є чутливими до зовнішніх впливів, що призводить до декогеренції, яка виникає внаслідок взаємодії з навколишнім середовищем і різними джерелами шуму. Такі ефекти зумовлюють появу помилок різної природи, зокрема фазових і бітових перевертань, а також їх комбінацій, що ускладнює забезпечення стабільності результатів обчислень. У зв'язку з цим для підвищення достовірності результатів застосовуються методи пом'якшення помилок (error mitigation techniques), які не виконують прямої корекції помилок, але дозволяють оцінити їхній вплив і частково компенсувати його на рівні статистичної обробки результатів вимірювань.

Перед виконанням квантової схеми алгоритму QASPA на фізичному пристрої вона проходить процес транспіляції з використанням Qiskit, який передбачає адаптацію логічної схеми до фізичної архітектури цільового квантового процесора. Процес транспіляції включає декілька послідовних етапів, спрямованих на узгодження структури квантового кола з обмеженнями апаратного забезпечення.

Першим етапом транспіляції є мапінг кубітів (qubit mapping), який визначає відображення логічних кубітів квантової схеми QASPA на фізичні кубіти квантового процесора. Даний етап є важливим, оскільки фізичні кубіти характеризуються обмеженою зв'язністю, і виконання двокубітних операцій можливе лише між безпосередньо з'єднаними кубітами. Транспілятор Qiskit використовує алгоритмічні підходи для вибору такого відображення, яке мінімізує необхідність застосування додаткових операцій маршрутизації. Зокрема, у випадках відсутності прямої зв'язності між кубітами виникає потреба у вставці SWAP-гейтів, що забезпечують переміщення квантової інформації між фізичними кубітами. Оскільки SWAP-гейт реалізується як послідовність трьох двокубітних операцій типу CNOT, його використання призводить до збільшення глибини квантової схеми та підвищення впливу шуму. У зв'язку з цим вибір ефективного мапінгу розглядається як один із

ключових чинників зменшення накопичення помилок під час виконання алгоритму.

Етап оптимізації та розкладу вентилів (gate decomposition and optimization) передбачає перетворення логічних вентилів алгоритму QASPA, зокрема гейт-триплету CX-Ry-CX, у базовий набір вентилів, що нативно підтримується апаратним забезпеченням. Це може включати розклад складних операцій на послідовності однокубітних обертань та двокубітних операцій типу CNOT. Транспілятор Qiskit також виконує оптимізацію порядку застосування вентилів з метою зменшення загальної глибини квантової схеми. Скорочення глибини є важливим чинником, оскільки тривалість когерентного існування кубітів обмежена, і збільшення часу виконання схеми підвищує вплив декогеренції та накопичення шуму.

Додатково, у разі підтримки відповідних механізмів на рівні апаратного забезпечення, може застосовуватися динамічна маршрутизація (dynamic routing), яка дозволяє адаптивно обирати шляхи виконання двокубітних операцій з урахуванням поточних характеристик кубітів і зв'язків. Це дає змогу уникати використання більш шумних елементів та підвищувати надійність виконання критичних операцій.

Після завершення транспіляції, але до фінального аналізу результатів, застосовуються методи пом'якшення помилок, які є важливою складовою експериментального протоколу на реальних квантових пристроях. Зокрема, для компенсації похибок зчитування виконуються калібрувальні вимірювання для відомих станів, на основі яких формується матриця помилок. Отримана інформація використовується для статистичної корекції результатів вимірювань алгоритму QASPA.

Екстраполяція нульового шуму (zero-noise extrapolation, ZNE) спрямована на оцінювання результатів, які могли б бути отримані за відсутності шуму, шляхом виконання квантової схеми з різними, штучно підвищеними рівнями шуму. Підвищення рівня шуму може досягатися, зокрема, за рахунок повторного застосування еквівалентних гейтів (що не

змінює логічної операції, але збільшує вплив шуму) або використання різних стратегій транспіляції, які призводять до зміни глибини схеми. Отримані результати для різних рівнів шуму піддаються подальшій екстраполяції до умов нульового шуму, що дозволяє отримати більш точну оцінку очікуваних значень. Інструменти, доступні в Qiskit Runtime та Qiskit Ignis, забезпечують практичну реалізацію підходу ZNE та сприяють зменшенню впливу фізичного шуму на результати алгоритму QASPA.

Стратегія динамічного розчеплення (dynamical decoupling, DD) передбачає застосування спеціальних послідовностей імпульсів до кубітів, які перебувають у стані очікування під час виконання квантової схеми. Такі імпульси дозволяють частково компенсувати вплив зовнішніх шумових факторів, сприяючи подовженню ефективного часу когерентності та зменшенню впливу декогеренції [140]. Застосування методів динамічного розчеплення дозволяє підвищити стабільність виконання алгоритму QASPA, особливо у випадках збільшеної глибини квантової схеми або при використанні процесорів із обмеженими когерентними характеристиками.

Після валідації алгоритму QASPA на локальному симуляторі наступним етапом є його виконання на реальному квантовому комп'ютері IBM. Такий підхід дозволяє дослідити поведінку алгоритму в умовах впливу шуму та апаратних обмежень, характерних для NISQ-ери. Для хмарного виконання використовується функція `run_qaspa_maxcut_cloud`, яка реалізує взаємодію з платформою IBM Quantum Platform через допоміжну функцію `run_on_cloud`, забезпечуючи повний цикл запуску квантових обчислень на віддаленому апаратному забезпеченні.

Лістинг функцій, що забезпечують запуск QASPA на хмарному квантовому комп'ютері:

```
def run_qaspa_maxcut_cloud(G, debug=False):

    graph = nx.to_numpy_array(G, dtype=int)
    n = len(graph)
    # Кількість запусків розраховується за моделлю масштабування (підрозділ 2.2)
    shots = int(np.ceil(0.03047 * (2 ** n) + 1016.2))
```

```

qc = qaspa_maxcut(graph)

counts = run_on_cloud(qc, debug, shots)

sorted_counts = sorted(counts.items(), key=lambda item: item[1], reverse=True)
top_4_results = []

for i, (bitstring, count) in enumerate(sorted_counts[:4]):
    solution = bitstring[::-1]
    cut_value = compute_cut_value(solution, graph)
    top_4_results.append({
        'rank': i + 1,
        'cut_value': cut_value,
        'solution': solution,
        'count': count
    })

# Best result is the first one
cloud_best_cut = top_4_results[0]['cut_value']
cloud_best_solution = top_4_results[0]['solution']

if debug:
    print("Final circuit:\n", qc)
    print("Counts:\n", dict(sorted(counts.items(), key=lambda item: item,
reverse=True)))
    print("Best solution:", cloud_best_solution[::-1])
    print("Best cut value:", cloud_best_cut)
    colors = ['r' if cloud_best_solution[node] == '0' else 'b' for node in G]
    nx.draw(G, with_labels=True, node_color=colors)
    plt.savefig("qaspa_maxcut_cloud.png")

return cloud_best_cut, cloud_best_solution

```

Функція `run_qaspa_maxcut_cloud` спочатку перетворює граф `G` у матрицю суміжності, потім, обчислюється оптимальна кількість `shots` для даного розміру графа. Квантова схема `qc` генерується за допомогою `qaspa_maxcut`, і потім вона передається до `run_on_cloud` для виконання. Після отримання результатів з хмари, функція ідентифікує найкращий розв’язок та його вартість розрізу, а також, у режимі відладки, може візуалізувати рішення.

```

from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2 as Sampler
from qiskit.transpiler import generate_preset_pass_manager

```

```

from qiskit.circuit import QuantumCircuit
from qiskit.visualization import plot_histogram
import numpy as np

def run_on_cloud(circuit, debug=False, shots=1024):

    service = QiskitRuntimeService(token='TOKEN', channel='ibm_cloud', instance='test')

    # Heron R2
    backend = service.backend('ibm_marrakesh')

    pm = generate_preset_pass_manager(optimization_level=1, backend=backend)
    isa_circuit = pm.run(circuit)
    param_values = np.random.rand(circuit.num_parameters) # Параметри для
    параметризованих схем, якщо такі є

    # execute the quantum circuit
    sampler = Sampler(backend)
    job = sampler.run([(isa_circuit, param_values)], shots=shots)
    result = job.result()
    pub_result = result
    counts = pub_result.data.meas.get_counts()
    if debug:
        print("Result: -----")
        print(result)
        print("-----")
        print("Counts: -----")
        print(counts)
        print("-----")
        plot_histogram(counts).savefig("ibm_result.png")
        print("File ibm_result.png saved")

    return counts

```

Функція `run_on_cloud` виступає інтерфейсом взаємодії з платформою IBM Quantum Platform та забезпечує ініціалізацію сервісу Qiskit Runtime. У межах цієї функції здійснюється вибір квантового бекенду, зокрема використано процесор `ibm_marrakesh` (архітектура IBM Heron, конфігурація R2), який характеризується зниженим рівнем шуму та покращеними експлуатаційними параметрами.

Ключовим етапом є транспіляція квантової схеми, яка виконується за допомогою методу `generate_preset_pass_manager` з рівнем оптимізації `optimization_level = 1`. На цьому етапі схема адаптується до топології зв'язності та базового набору вентилів обраного квантового процесора. Результатом транспіляції є схема `isa_circuit` (рис. 3.7), яка приведена до апаратно-сумісного вигляду та готова до виконання. Для запуску квантової схеми використовується примітив `SamplerV2`, який забезпечує виконання схеми на квантовому пристрої та збір статистики результатів вимірювань. Отриманий об'єкт `PrimitiveResult` містить дані вимірювань, на основі яких формується розподіл бітових рядків (`counts`), що використовується для подальшого аналізу результатів алгоритму QASPA.

Результати виконання алгоритму QASPA на реальному квантовому процесорі IBM Marrakesh (156 кубітів) для тестового графа наведено нижче:

Result: -----

```
PrimitiveResult([SamplerPubResult(data=DataBin(c=BitArray(<shape=(),
num_shots=1024, num_bits=5>), meas=BitArray(<shape=(), num_shots=1024,
num_bits=5>)), metadata={'circuit_metadata': {} }]), metadata={'execution':
{'execution_spans': { '__type__': 'ExecutionSpans', '__value__': {'spans':
[DoubleSliceSpan(<start='2025-11-18 07:35:22', stop='2025-11-18 07:35:27',
size=1024>)] } }, 'version': 2})
```

Counts: -----

```
{'10101': 141, '01000': 7, '10100': 60, '11010': 160, '11111': 2, '10010': 20,
'01010': 161, '01001': 14, '11100': 16, '00101': 149, '11000': 22, '01101': 8, '10001':
51, '00110': 21, '00111': 18, '01100': 13, '10110': 19, '01011': 24, '00011': 16, '10011':
21, '00001': 11, '10111': 3, '11001': 13, '01110': 26, '10000': 4, '00100': 10, '11110':
2, '01111': 1, '11011': 7, '00000': 1, '00010': 3}
```

File ibm_result.png saved

Counts:

{'01010': 161, '11010': 160, '00101': 149, '10101': 141, '10100': 60, '10001': 51, '01110': 26, '01011': 24, '11000': 22, '00110': 21, '10011': 21, '10010': 20, '10110': 19, '00111': 18, '11100': 16, '00011': 16, '01001': 14, '01100': 13, '11001': 13, '00001': 11, '00100': 10, '01101': 8, '01000': 7, '11011': 7, '10000': 4, '10111': 3, '00010': 3, '11111': 2, '11110': 2, '01111': 1, '00000': 1}

Best solution: 01010

Best cut value: 6

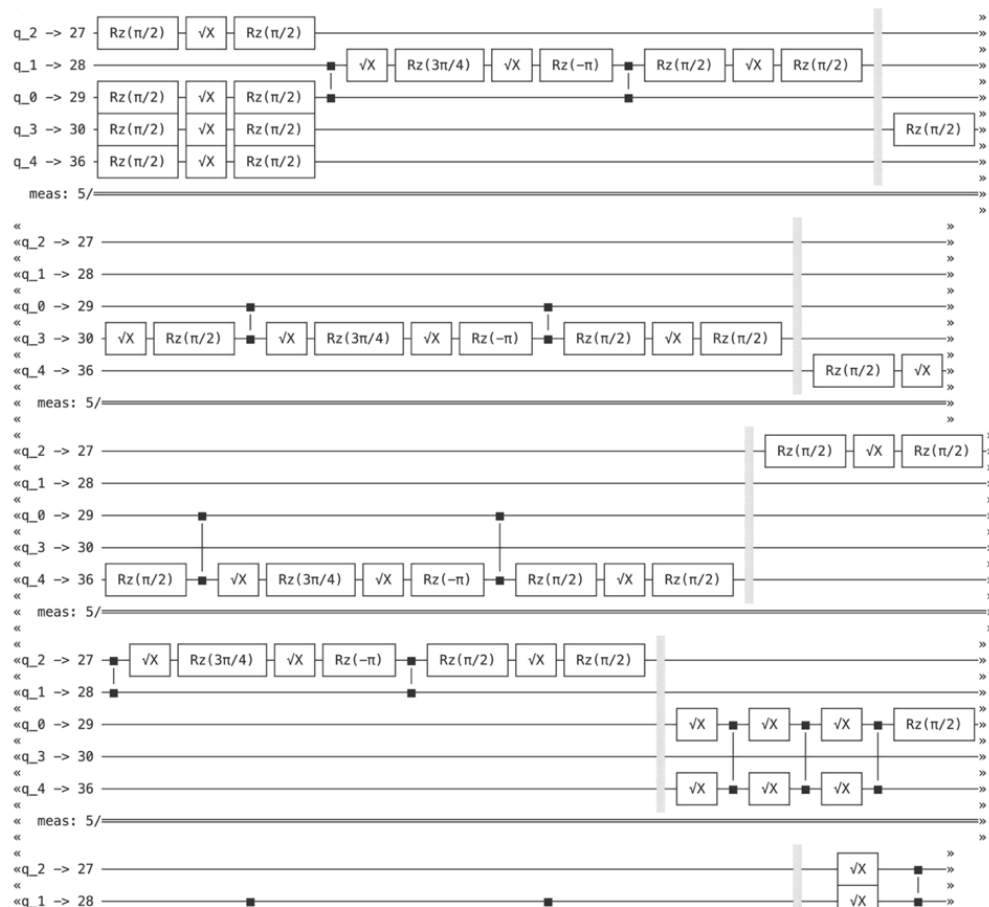


Рис. 3.7. Оптимізована квантова схема алгоритму QASPA

З цих даних видно, що найкращим розв'язком, отриманим з реального квантового комп'ютера (рис. 3.8), є бітовий рядок 01010 з частотою 161. Вартість розрізу для цього розв'язку також становить 6, що збігається з оптимальним значенням, знайденим на симуляторі. Інші домінуючі розв'язки включають 11010 (160 разів), 00101 (149 разів) та 10101 (141 раз).

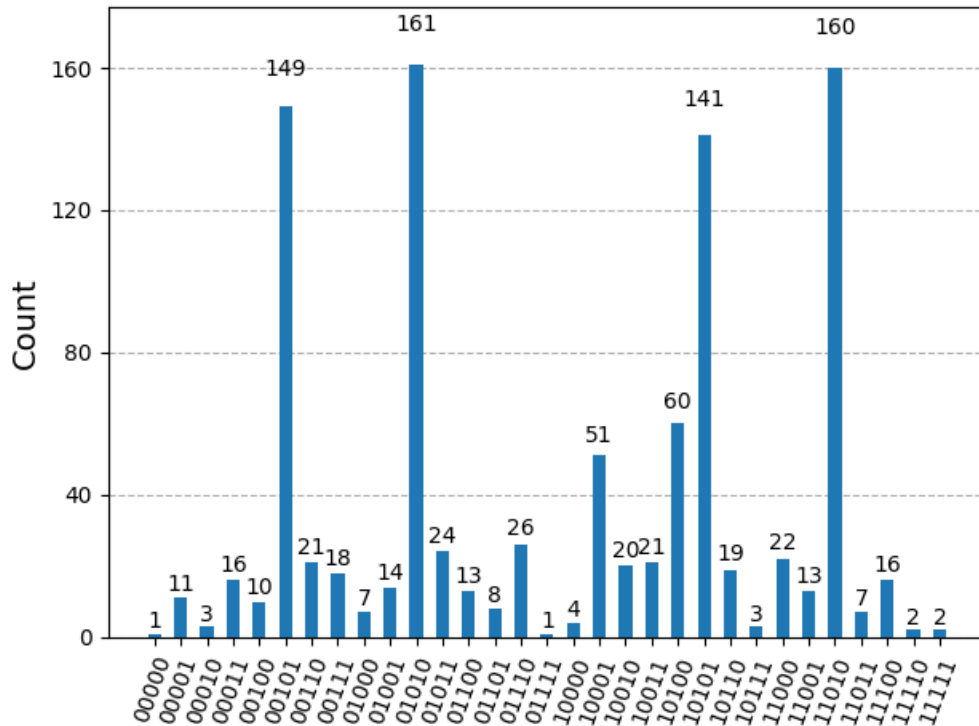


Рис. 3.8. Гістограма розподілу результатів вимірювань алгоритму QASPA на реальному квантовому комп'ютері

Порівняння гістограм, отриманих у результаті виконання алгоритму на локальному симуляторі та реальному квантовому комп'ютері, демонструє як подібність у домінуючих розв'язках, так і відмінності, зумовлені впливом шуму. Зокрема, для реального квантового пристрою характерною є менша вираженість піків та більш рівномірний розподіл ймовірностей, що пояснюється ефектами декогеренції та апаратних похибок. Водночас ключові оптимальні розв'язки залишаються ідентифікованими серед отриманих результатів, що свідчить про здатність алгоритму QASPA зберігати ефективність інтерференційного підсилення навіть в умовах NISQ-обмежень.

У проведеному експерименті для хмарного запуску найбільш імовірним виявився бітовий рядок 01010, тоді як у локальному моделюванні – 00101. Обидві конфігурації відповідають максимальному значенню розрізу, що дорівнює 6 для розглянутого графа. Це узгоджується з теоретичними властивостями задачі Max-Cut та свідчить про стійкість алгоритму до варіацій результатів вимірювань.

Візуалізація тестового графа з розфарбуванням вершин відповідно до знайденого розбиття (01010) наведена на рис. 3.9, де різні кольори позначають належність вершин до підмножин розрізу, а ребра, що їх з'єднують, відображають структуру отриманого рішення.

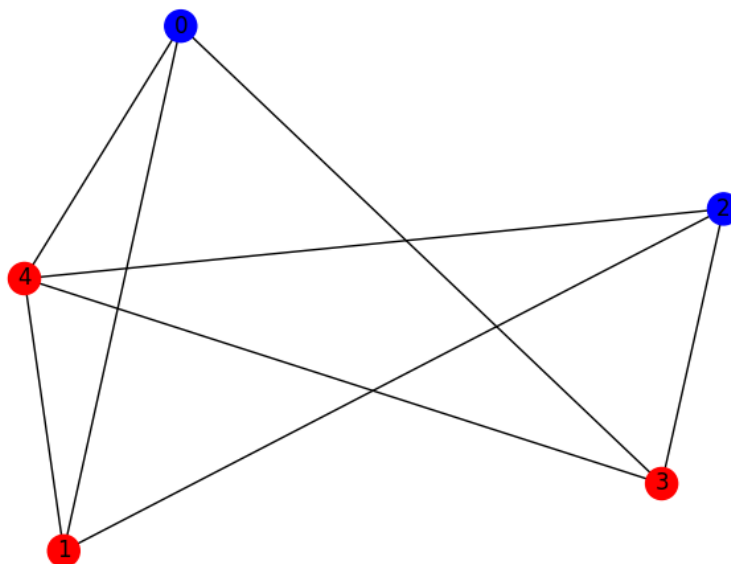


Рис. 3.9. Тестовий граф із візуалізацією знайденого розрізу 01010 на основі результатів реального квантового пристрою

Порівняння результатів виконання алгоритму QASPA на локальному симуляторі та реальному квантовому комп'ютері надає можливість оцінити його продуктивність і стійкість у різних обчислювальних середовищах. Насамперед слід зазначити, що в обох випадках – як на симуляторі, так і на реальному квантовому пристрої – алгоритм QASPA ідентифікує розв'язки, що відповідають максимальному розрізу тестового графа, який становить 6. Це узгоджується з теоретичними властивостями алгоритму та свідчить про коректність реалізованих механізмів квантового підсилення, зокрема фазових перетворень навколо осі O_y та кодування ваг ребер у параметризованій квантовій схемі. Отримані результати демонструють, що зазначені механізми зберігають свою ефективність навіть за наявності фізичних шумів.

Водночас спостерігаються відмінності у розподілі частот бітових рядків між симулятором і реальним квантовим пристроєм. Для симулятора, де

відсутній фізичний шум, розподіл counts характеризується чітко вираженими домінуючими піками, що значно перевищують інші конфігурації. Натомість на реальному квантовому комп'ютері розподіл є більш згладженим, а частоти найбільш імовірних розв'язків зменшуються (наприклад, 212 для 00101 на симуляторі проти 149 на апаратному виконанні; 167 для 01010 проти 161 відповідно). Такі відмінності є наслідком впливу фізичного шуму, декогеренції та похибок виконання гейтів, характерних для NISQ-пристроїв. Зазначені фактори призводять до зменшення ймовірності вимірювання оптимальних станів і водночас до збільшення частки шумових конфігурацій у розподілі результатів.

Незважаючи на вплив шуму, важливо відзначити, що домінуючі розв'язки, отримані на реальному квантовому пристрої, відповідають оптимальним конфігураціям задачі. Це свідчить про робастність алгоритму QASPA та його придатність до застосування в умовах NISQ-ери. Застосовані методи пом'якшення помилок, інтегровані у Qiskit Runtime, ймовірно, сприяють підвищенню якості отриманих результатів. Порівняльний аналіз гістограм дозволяє оцінити ефективність застосованих методів пом'якшення помилок, а також ідентифікувати компоненти квантової схеми QASPA, найбільш чутливі до впливу шуму.

Отримані результати на реальних квантових пристроях свідчать про практичну реалізованість алгоритму QASPA. Вони демонструють, що запропонований одноітераційний квантово-гібридний підхід здатний знаходити високоякісні розв'язки задачі Max-Cut не лише в ідеалізованих симуляційних умовах, але й у середовищі сучасних шумних квантових пристроїв. Таким чином, результати експериментального дослідження узгоджуються з теоретичними положеннями та підтверджують доцільність подальшого розвитку підходу QASPA для розв'язання задач комбінаторної оптимізації в умовах обмежених ресурсів квантових обчислювальних систем.

3.3. Тестування та порівняння з іншими алгоритмами для розв’язання Max-Cut

Для всебічної валідації ефективності та надійності розробленого квантово-гібридного алгоритму QASPA, а також для оцінювання його продуктивності у порівнянні з іншими підходами до розв’язання задачі Max-Cut, було проведено серію експериментальних досліджень. Зазначений етап дозволяє перевірити теоретичні властивості алгоритму, розглянуті в розділі 2, на практичних прикладах з урахуванням обмежень, характерних для NISQ-ери. Порівняльний аналіз включав зіставлення результатів алгоритму QASPA з точним розв’язком, отриманим методом повного перебору, а також з результатами реалізації квантового наближеного оптимізаційного алгоритму (QAOA), зокрема із застосуванням фреймворку Qrisp.

Для забезпечення коректності порівняння було сформовано набір тестових графів, який охоплює різні топологічні структури та розміри, характерні для задач комбінаторної оптимізації невеликої та середньої складності. Розглянуті графи включають конфігурації з кількістю вершин від 3 до 7, зокрема лінійні структури, циклічні графи, трикутники, а також графи з підвищеною щільністю зв’язків.

Першим етапом порівняльного аналізу стала реалізація методу повного перебору (brute-force algorithm) для задачі Max-Cut. Даний підхід, незважаючи на експоненційну обчислювальну складність, є придатним для графів невеликої розмірності та забезпечує знаходження оптимального розв’язку шляхом систематичного перебору всіх можливих розбиттів множини вершин.

Для графа з n вершинами кількість унікальних розбиттів становить 2^{n-1} що зумовлює обмеження застосування методу для задач більшої розмірності. Водночас результати, отримані за допомогою повного перебору, використовуються як еталонні значення для оцінювання якості розв’язків, отриманих іншими алгоритмами.

Реалізація функції `brute_force` була наведена у попередніх підрозділах. Функція приймає на вхід граф G , який перетворюється у матрицю суміжності, після чого здійснюється генерація всіх можливих бітових рядків довжини `num_nodes`, що відповідають різним варіантам розбиття вершин. Для кожного розбиття обчислюється значення розрізу за допомогою функції `compute_cut_value`, а найкращий результат зберігається як оптимальний. Таким чином, метод повного перебору використовується як базова контрольна процедура для перевірки коректності результатів, отриманих алгоритмом QASPA та іншими підходами, у межах розглянутих тестових прикладів.

Для порівняння алгоритму QASPA з іншим варіаційним квантовим підходом було реалізовано стандартну версію алгоритму QAOA із використанням фреймворку Qrisp. Qrisp є квантовим фреймворком, який інтегрується з Qiskit та спрощує розробку і виконання квантових алгоритмів на реальних квантових комп'ютерах, зокрема QAOA [106]. Як було детально описано в підрозділі 1.3, QAOA є гібридним квантово-класичним алгоритмом, призначеним для розв'язання задач комбінаторної оптимізації в умовах NISQ-ери [80]. Алгоритм використовує параметризовану квантову схему та класичний оптимізатор для пошуку параметрів, що максимізують імовірність отримання розв'язку. Реалізацію QAOA із використанням Qrisp було сформовано на основі офіційного прикладу, наведеного в документації фреймворку [69]. У цій реалізації `QAOAProblem` автоматично формує оператор вартості на основі графа G та використовує стандартний міксерний оператор `RX_mixer`. Кількість ітерацій класичного оптимізатора (`max_iter`) було встановлено рівною 10 (з урахуванням того, що алгоритм QASPA використовує один запуск без ітерацій), що є достатнім для графів невеликої розмірності, проте може бути недостатнім для складніших ландшафтів функції вартості. Результат `res` подається у вигляді словника, що містить бітові рядки та відповідні ймовірності, серед яких як розв'язок обирається найбільш імовірна конфігурація.

Для систематичної оцінки QASPA, QAOA та методу повного перебору було розроблено набір із 17 тестових випадків (test1 до test17). Кожен тестовий випадок ініціалізує унікальний граф G з різною кількістю вершин (від 3 до 7) та конфігураціями ребер. Ці графи включають прості структури, такі як цикли, шляхи, а також більш складні, але невеликі графи, які дозволяють однозначно визначити оптимальне рішення за допомогою повного перебору. Для кожного тестового графа виконувалися наступні дії:

- локальний запуск QASPA: `run_qaspa_maxcut_local(G)`;
- запуск QAOA: `qaoa_qrisp(G)`;
- запуск повного перебору: `brute_force(G)`.

Була додана функція `run_single_manual_test` яка запускала алгоритми на тестових графах та перевіряла правильність їх роботи:

```
def run_single_manual_test(test_name, debug=False):
    sample = get_test_sample(test_name)
    if sample is None:
        print(f'Test '{test_name}' not found!')
        return
    G = sample.build_graph()

    # Run all algorithms
    (qaspa_cut, qaspa_solution, _) = run_qaspa_maxcut_local(
        G, debug=debug, angle=np.pi/4, deep=1)
    (brute_force_cut, brute_force_solution) = brute_force(G)
    (qaoa_cut, qaoa_solution) = qaoa_qrisp(G, debug=debug)

    try:
        assertEquals(test_name + " QASPA", qaspa_cut, brute_force_cut, )
        assertEquals(test_name + " QAOA", qaoa_cut, brute_force_cut, )
        print(f'\n')
    except AssertionError as e:
        ratio = qaspa_cut / brute_force_cut if brute_force_cut > 0 else 0
        print(f'\nX {e} (Ratio: {ratio:.4f})')

    return qaspa_cut, brute_force_cut, qaoa_cut
```

Інші тестові кейси описані у додатку А.

Кожен тест передбачає функціональне порівняння результату виконання алгоритму QASPA (`run_qaspa_maxcut_local`) з еталонним значенням `brute_force_best_cut`. Аналогічні перевірки проводилися для хмарного виконання (`run_qaspa_maxcut_cloud`), а також для реалізації QAOA (`qaoa_qrisp`).

За результатами проведених експериментів алгоритм QASPA продемонстрував високу стабільність і точність. Усі 17 тестових випадків, що охоплювали граfi з кількістю вершин від 3 до 7 та різними топологічними структурами, були коректно розв'язані алгоритмом як у режимі локальної симуляції, так і при виконанні на реальних квантових пристроях IBM. У кожному випадку значення максимального розрізу, отримане за допомогою QASPA, збігалось з оптимальним значенням, визначеним методом повного перебору. Отримані результати свідчать про коректність програмної реалізації алгоритму та підтверджують його здатність знаходити оптимальні розв'язки задачі Max-Cut для графів малої та середньої розмірності.

Таблиця 3.1

Результати порівняльного тестування алгоритмів

№	QASPA Cut	QASPA Solution	QAOA Cut	Brute Force Cut
1	6	[1' 0' 1' 0' 0']	6	6
2	2	[1' 1' 0']	2	2
3	4	[0' 1' 0' 1']	4	4
4	4	[1' 0' 1' 0']	2	4
5	4	[1' 0' 1' 0']	4	4
6	4	[0' 1' 0' 0' 1']	4	4
7	3	[1' 1' 0' 0']	3	3
8	5	[0' 0' 1' 0' 1' 1']	5	5
9	3	[1' 0' 1' 1']	3	3
10	5	[1' 0' 1' 0' 0']	4	5
11	6	[1' 0' 1' 0' 1' 0']	6	6
12	3	[0' 1' 0' 1']	3	3
13	6	[0' 1' 0' 1' 0' 1']	6	6
14	6	[1' 0' 1' 0' 1' 1']	6	6
15	6	[1' 0' 0' 0' 1' 0']	4	6
16	6	[0' 1' 0' 1' 0' 1']	6	6
17	10	[1' 0' 1' 1' 1' 0' 0']	10	10

Для ілюстрації порівняльної продуктивності алгоритмів QASPA, QAOA та методу повного перебору наведено зведену таблицю 3.1, що містить результати для 17 тестових випадків. Отримані результати демонструють ефективність алгоритму QASPA при виконанні як на симуляторі, так і на реальних квантових пристроях, незважаючи на вплив шуму, декогеренції та обмежень зв'язності, характерних для NISQ-комп'ютерів. Стабільність результатів у таких умовах свідчить про доцільність використання запропонованої одноітераційної моделі, у межах якої параметри квантової схеми визначаються безпосередньо на основі структури графа, як це детально описано в підрозділі 2.3.

Застосована параметризація, що поєднує фазові зсуви навколо осі O_y у міксерних операторах із кодуванням зв'язків ребер, забезпечує підсилення оптимальних конфігурацій і водночас зменшує залежність від класичного оптимізаційного циклу, який є чутливим до шуму. Додатково, використання методів пом'якшення помилок на платформі IBM Quantum Platform, як зазначено в підрозділі 3.2, сприяє підвищенню якості результатів, отриманих на реальному квантовому обладнанні.

На відміну від алгоритму QASPA, реалізація QAOA із використанням фреймворку Qrisp у серії експериментів продемонструвала загалом високу точність, однак не у всіх тестових випадках досягала оптимального значення, визначеного методом повного перебору. Відхилення від оптимуму спостерігалися у тестах 4, 10 та 15 (рис. 3.10).

Зокрема, у тесті 4 для графа з 5 вершинами та 4 ребрами алгоритм QAOA отримав значення розрізу 2 при оптимальному значенні 4 (коефіцієнт апроксимації 0.5). У тесті 10 (5 вершин, 6 ребер) було досягнуто значення 4 за оптимального значення 5 (коефіцієнт апроксимації 0.8). Аналогічно, у тесті 15 (6 вершин, 6 ребер) отримано значення 4 замість оптимального 6 (коефіцієнт апроксимації 0.67). В інших тестових випадках результати QAOA збігалися з оптимальними значеннями, що свідчить про його ефективність для частини

розглянутих задач, проте вказує на наявність залежності якості розв’язку від конкретної структури графа та параметрів оптимізації.

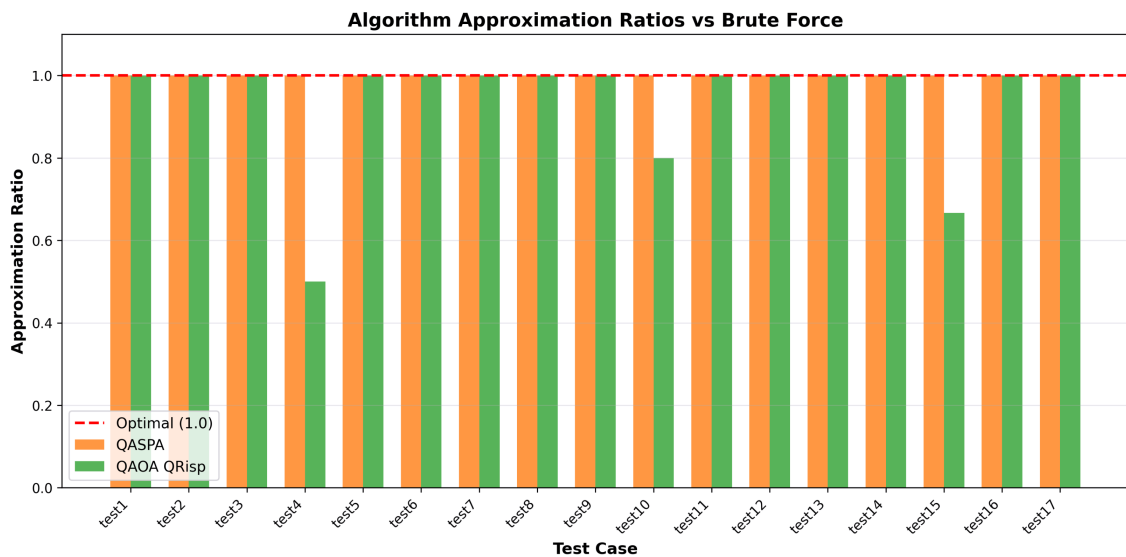


Рис. 3.10. Гістограма рівня апроксимації у проведених тестах

Отримані результати свідчать про наукову новизну алгоритму QASPA, яка полягає у запропонованому квантово-гібридному підході, що поєднує фазові зсуви навколо осі O_y у міксерних операторах із кодуванням ваг ребер графа. Така побудова дозволяє підвищити якість наближених розв’язків у порівнянні з базовою реалізацією QAOA для графів малої та середньої розмірності. Запропонований підхід до параметризації варіаційних квантових схем, який враховує структуру графа та локальні характеристики його вершин, забезпечує адаптивність алгоритму та стабільність результатів у досліджених експериментальних сценаріях. Це, у свою чергу, дозволяє алгоритму QASPA знаходити оптимальні розв’язки для розглянутих класів задач, що є важливим з точки зору практичного застосування.

Таким чином, проведений порівняльний аналіз свідчить про ефективність та надійність алгоритму QASPA для розв’язання задачі Max-Cut на графах малої та середньої розмірності. Здатність алгоритму стабільно знаходити оптимальні розв’язки як у симуляційному середовищі, так і на реальних квантових пристроях, у поєднанні з меншою залежністю від класичного оптимізаційного циклу порівняно з QAOA, вказує на

перспективність подальшого розвитку запропонованого підходу в межах квантово-гібридної оптимізації.

Для більш глибокого аналізу масштабованості та порівняльної стійкості алгоритмів QASPA і QAOA було реалізовано систему експериментального тестування на згенерованих графах різних типів і розмірностей. Такий підхід дозволяє дослідити поведінку алгоритмів за різних структурних характеристик задачі та оцінити їхню ефективність у широкому спектрі конфігурацій. У межах експериментів використовувалися кілька класів графів: повні (Complete Graph), циклічні (Cycle Graph), а також випадкові графи моделі Erdős-Rényi з різними ймовірностями утворення ребра ($(p = 0.3)$, $(p = 0.5)$, $(p = 0.7)$). Такий вибір забезпечує варіювання щільності зв'язків – від регулярних структур до стохастичних графів із різним рівнем насиченості ребрами, що формує різні за складністю варіанти задачі Max-Cut.

Розмірність графів змінювалася в діапазоні від 3 до 14 вершин, що дозволило проаналізувати вплив масштабу задачі на якість отриманих розв'язків. Для кожної комбінації типу графа та кількості вершин виконувалося по три незалежні запуски (trials), що підвищує статистичну надійність результатів. Критерієм успішності виступав коефіцієнт апроксимації не менше 0.9, де оптимальне значення визначалося методом повного перебору. Кожен експериментальний запуск передбачав порівняння отриманого значення розрізу з еталонним оптимумом, що забезпечувало уніфікований критерій оцінювання якості результатів. Організація обчислювального процесу була реалізована у вигляді вкладеної ітераційної структури, яка забезпечує систематичний перебір розмірностей графів, типів топологій та номерів повторів. Такий підхід гарантує повне покриття всіх комбінацій параметрів тестування та формує узгоджену статистичну вибірку.

Загальна логіка експериментальної процедури наведена у фрагменті програмного коду нижче, де послідовно здійснюється перебір кількості вершин, генерація графів різних типів і запуск процедури оцінювання для кожного з них (повний лістинг наведено у додатку A):

```

for graph_size in GRAPH_SIZES:
    print(f"\nTesting graphs with {graph_size} nodes...")

    for trial_num in range(num_trials):
        graphs = generate_graphs(graph_size, trial_num)

        for graph_type, G in graphs.items():
            current_test += 1

            # Progress indicator
            progress = f"[{current_test:3d}/{total_tests}]"
            type_info = f"{graph_type:20s} Size={graph_size:2d}"
            Trial=f"{trial_num+1}/{num_trials}"
            size_info = f"Edges={G.number_of_edges():3d}"
            print(f" {progress} {type_info} {size_info}", end=" ... ", flush=True)

            # Run test
            result = self.run_test_for_graph(G, graph_type, graph_size, trial_num)

            if result and result.bf_cut is not None:
                qaspa_info = f"QASPA: {result.qaspa_ratio:.3f}" if result.qaspa_ratio else
                "QASPA: ERROR"
                qaoa_info = f"QAOA: {result.qaoa_ratio:.3f}" if result.qaoa_ratio else "QAOA:
                ERROR"
                print(f"{qaspa_info}, {qaoa_info}")
            else:
                print("FAILED")

```

У наведеному фрагменті програмного коду зовнішній цикл забезпечує послідовне збільшення розмірності графів відповідно до множини GRAPH_SIZES. Для кожного значення кількості вершин виконується декілька незалежних запусків (num_trials), що дозволяє зменшити вплив стохастичних факторів, пов'язаних із генерацією графів та процесом оптимізації. Функція generate_graphs формує набір графів різних типів однакової розмірності, після чого для кожного з них виконується уніфікована процедура run_test_for_graph, яка обчислює значення розрізу для алгоритмів QASPA, QAOA та еталонного методу повного перебору.

Отримані коефіцієнти апроксимації (qaspa_ratio, qaoa_ratio) автоматично виводяться разом зі службовою інформацією про хід

експерименту, включаючи тип графа, його розмірність та кількість ребер. Така організація експериментального процесу забезпечує відтворюваність результатів, контроль коректності виконання та створює передумови для подальшого статистичного аналізу. Застосування описаної системи тестування дозволило отримати репрезентативні дані щодо поведінки алгоритмів у різних топологічних умовах та при зростанні розмірності задачі.

Результати експериментів (рис. 3.11) на графах різних типів (Complete, Cycle, Erdős–Rényi з $p = 0.3, 0.5, 0.7$) та розмірів від 3 до 14 вершин дозволили сформулювати узагальнену оцінку масштабованості та стабільності алгоритмів QASPA і QAOA. Оцінювання проводилося за двома ключовими метриками: середнім коефіцієнтом апроксимації (mean approximation ratio) та часткою успішних запусків (success rate) за умови досягнення порогу $ratio \geq 0.9$.

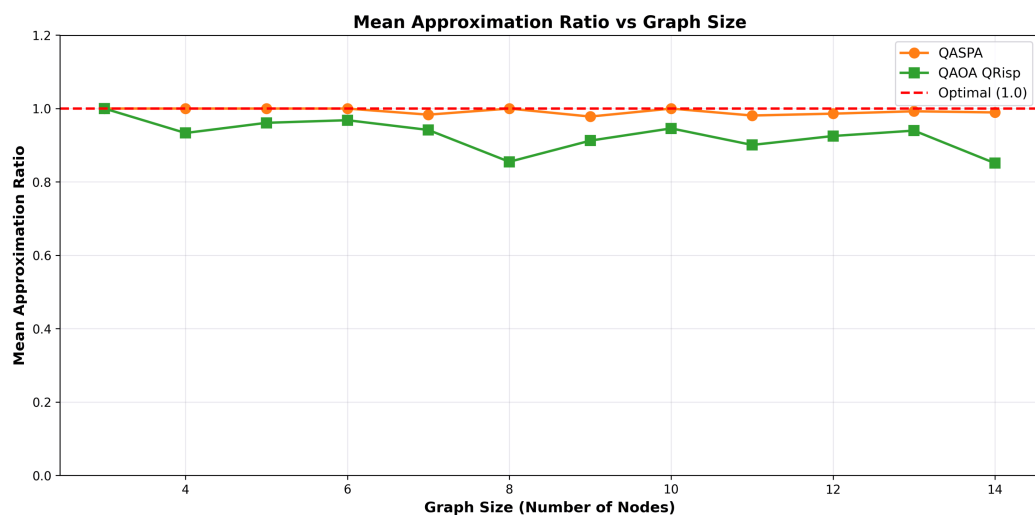


Рис. 3.11. Графік рівня апроксимації відносно кількості вершин

Як показано на гістограмі (рис. 3.11), алгоритм QASPA демонструє середній коефіцієнт апроксимації, близький до 1.0, для всіх розмірів графів у досліджуваному діапазоні. Спостережувані коливання є незначними та не виявляють тенденції до систематичного зниження зі зростанням кількості вершин. Для алгоритму QAOA, навпаки, спостерігається більша варіативність результатів. При малих розмірах графів (3-6 вершин) значення коефіцієнта апроксимації залишаються близькими до оптимальних, однак починаючи з 7-8 вершин фіксується тенденція до зниження середнього показника з окремими

локальними мінімумами. Для графів максимальної розмірності (14 вершин) середній коефіцієнт апроксимації QAOA зменшується до приблизно 0.85, що може свідчити про зниження стабільності алгоритму при зростанні складності задачі.

Аналіз гістограми залежності частки успішних запусків (success rate) від розміру графа (рис. 3.12) показує, що алгоритм QASPA демонструє 100% успішність для всіх значень n у досліджуваному діапазоні.

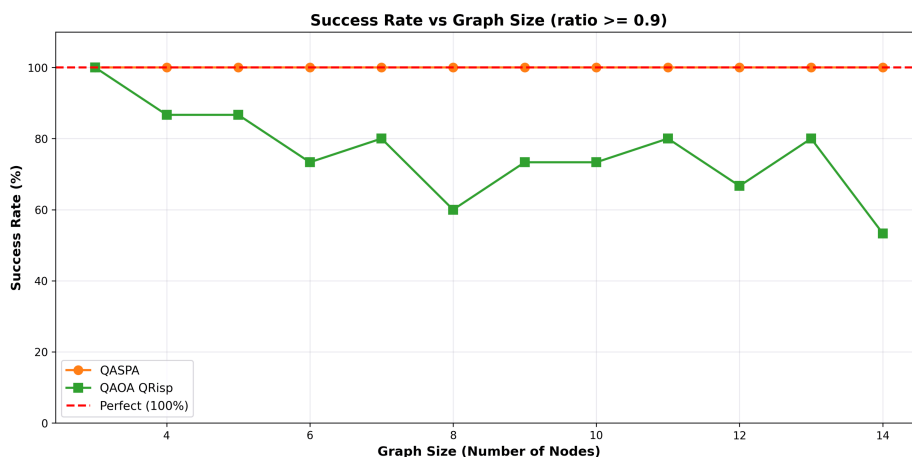


Рис. 3.12. Графік частки успішних запусків відносно кількості вершин

Для алгоритму QAOA спостерігається зниження частки успішних запусків зі збільшенням розміру графа: починаючи з 6-7 вершин показник зменшується до 60–80%, а для графів із 14 вершинами становить приблизно 53%. Така динаміка може бути пов'язана зі зростанням складності оптимізаційного ландшафту у просторі параметрів (γ, β) при масштабуванні задачі, що ускладнює ефективний пошук оптимальних значень класичним оптимізатором.

Порівняння результатів за типами графів (рис. 3.13) додатково ілюструє залежність ефективності алгоритму QAOA від топологічних характеристик задачі. Згідно з отриманими даними, алгоритм QASPA зберігає 100% рівень успішності для всіх розглянутих класів графів. Для QAOA найвищий показник успішності (100%) спостерігається на циклічних графах, тоді як для повних графів і графів Erdős-Rényi model з високою щільністю ребер частка успішних запусків знижується до 50-86%.

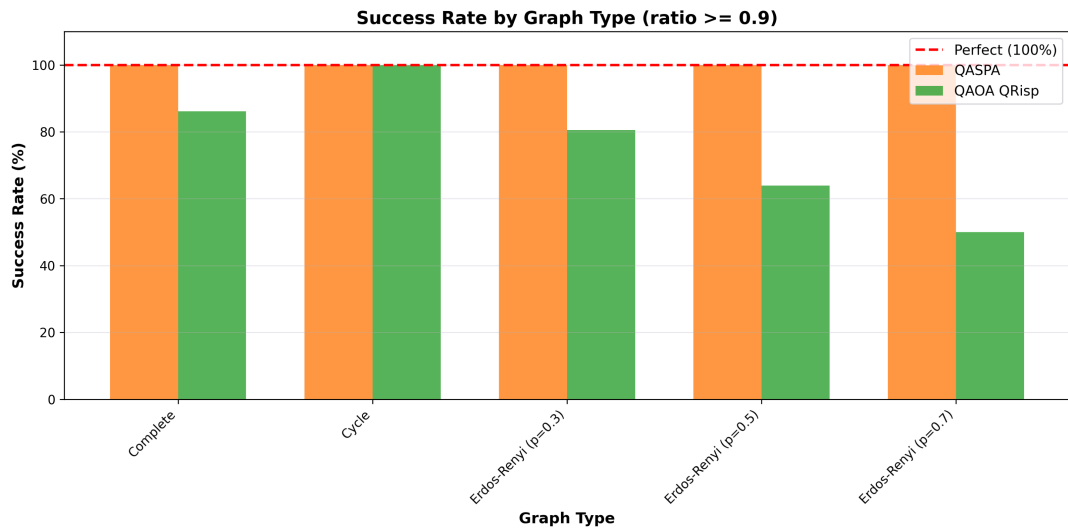


Рис. 3.13. Гістограма частки успішних запусків відносно типу графів

Така залежність може бути пояснена зростанням щільності графа та відповідним ускладненням функції вартості, що підвищує ймовірність потрапляння класичного оптимізатора в локальні мінімуми. Окремого аналізу потребує залежність коефіцієнта апроксимації від типу графа, оскільки саме топологічні характеристики визначають структуру функції вартості задачі Max-Cut і складність параметричного ландшафту варіаційного алгоритму.

Як показано на гістограмі середнього коефіцієнта апроксимації за типами графів (рис. 3.14), алгоритм QASPA демонструє практично однакові значення, близькі до 1.0, для всіх розглянутих топологій: повних графів, циклічних графів та випадкових графів Erdős-Rényi model з параметрами $p=0.3$, $p=0.5$ та $p=0.7$.

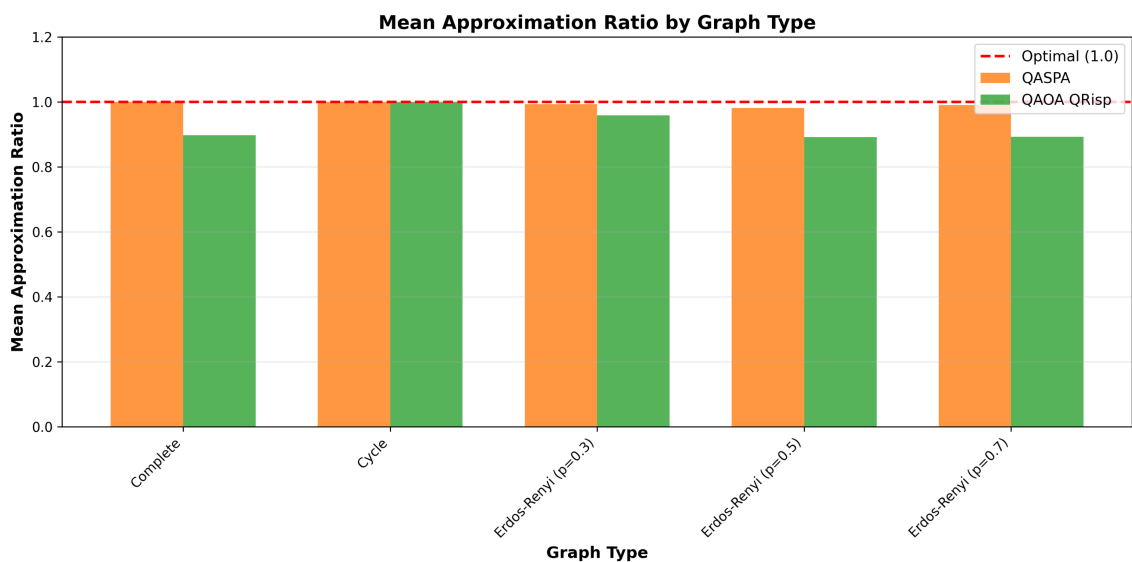


Рис. 3.14. Гістограма рівня апроксимації відносно типу графів

Відсутність статистично значущих відхилень може свідчити про топологічну інваріантність запропонованої параметризації квантової схеми. Інакше кажучи, ефективність QASPA не демонструє суттєвої чутливості до зміни щільності або регулярності структури графа в межах досліджуваного діапазону.

Для алгоритму QAOA спостерігається інша тенденція. Найвищий середній коефіцієнт апроксимації зафіксовано для циклічних графів, де він наближається до оптимального значення (близько 1.0), що може бути пояснено регулярністю структури та відносною простотою оптимізаційного ландшафту для таких графів. Для повних графів середній коефіцієнт апроксимації знижується до приблизно 0.89-0.90, що може вказувати на зростання складності функції вартості внаслідок високої щільності зв'язків.

Особливо показовою є поведінка QAOA на випадкових графах Erdős-Rényi. При $p=0.3$ середній коефіцієнт апроксимації залишається відносно високим (близько 0.95), а success rate перевищує 90%, однак зі збільшенням імовірності зв'язку до $p=0.5$ та $p=0.7$ спостерігається зниження показника до приблизно 0.63. При цьому success rate демонструє ще більш виражену деградацію, зменшуючись до рівня 60-50% для графів більшої розмірності. Це свідчить про те, що підвищення щільності графа у стохастичних структурах ускладнює оптимізаційний процес, збільшує розмірність простору пошуку та підвищує ймовірність потрапляння класичного оптимізатора в локальні мінімуми.

Таким чином, аналіз залежності коефіцієнта апроксимації від типу графа підтверджує структурну стабільність алгоритму QASPA та виявляє топологічну залежність ефективності QAOA. Зростання щільності з'єднань і стохастичності структури безпосередньо впливає на якість розв'язків QAOA, тоді як QASPA зберігає стабільний рівень апроксимації в межах дослідженого діапазону розмірностей.

Для всебічної оцінки розробленого квантово-гібридного алгоритму QASPA-MaxCut, а також об'єктивного визначення його переваг та обмежень у контексті сучасних підходів до розв'язання задачі максимального розрізу, було проведено аналіз продуктивності за низкою ключових критеріїв. Зазначені критерії охоплюють основні характеристики, що визначають ефективність та практичну придатність квантових алгоритмів в умовах NISQ-ери.

Якість розв'язку є одним із ключових показників ефективності оптимізаційного алгоритму та визначається його здатністю знаходити оптимальне або близьке до оптимального значення цільової функції (вартості розрізу). Проведені експериментальні дослідження алгоритму QASPA демонструють його високу здатність до знаходження оптимальних розв'язків. Зокрема, алгоритм коректно визначив максимальний розріз для всіх 17 тестових випадків з різною топологією графів (від 3 до 7 вершин), а також для 65 тестових випадків при дослідженні циклічних, повних та випадкових графів забезпечив отримання оптимальних або близьких до оптимальних значень. У кожному з розглянутих випадків значення вартості розрізу, отримане за допомогою QASPA, або збігалось з еталонним значенням, або досягало його з коефіцієнтом апроксимації $p > 0.9$, визначеним на основі методу повного перебору, що гарантує знаходження глобального оптимуму. Отримані результати свідчать про стабільно високу якість розв'язків, що забезпечується запропонованою параметризацією квантової схеми, яка поєднує фазове кодування на осі O_y із урахуванням ваг ребер графа. Такий підхід сприяє підсиленню перспективних конфігурацій і підвищує ймовірність отримання оптимальних розв'язків при вимірюванні.

Швидкість збіжності варіаційних квантових алгоритмів зазвичай оцінюється кількістю ітерацій класичного оптимізатора, необхідних для досягнення стаціонарного значення енергії. У цьому аспекті алгоритм QASPA демонструє суттєву перевагу завдяки одноітераційній організації обчислювального процесу, як це детально описано в підрозділі 2.1. На відміну

від класичного QAOA, який передбачає багаторазовий ітераційний цикл взаємодії між квантовим комп'ютером та класичним оптимізатором для пошуку оптимальних параметрів, у QASPA-MaxCut параметри квантової схеми визначаються детерміновано на основі структури графа. Зокрема, кути обертання обчислюються класично з використанням матриці суміжності та передаються до квантової схеми без необхідності подальшої ітераційної корекції. Таким чином, у рамках запропонованого підходу відсутній ітераційний етап класичної оптимізації, що дозволяє розглядати процес отримання розв'язку як одноетапний. Це, у свою чергу, усуває залежність від складного ландшафту функції вартості та зменшує ризик потрапляння у локальні мінімуми, що є характерною проблемою для варіаційних алгоритмів. У результаті, запропонована архітектура забезпечує скорочення загального часу обчислення та підвищує ефективність алгоритму з точки зору обчислювальних витрат.

Глибина квантового кола, що визначається кількістю вентилів і шарів, є критичним параметром для NISQ-пристроїв, оскільки вона безпосередньо впливає на час когерентності та рівень накопичення помилок. Алгоритм QASPA-MaxCut розроблений з урахуванням цих обмежень. Як детально описано в підрозділі 2.3, QASPA використовує стандартизований тригейтовий блок CX-Ry-CX для кодування ребер графа. Така конструкція є апаратно-сумісною реалізацією керованого обертання, що відображає зв'язки між вершинами у вигляді локальних амплітудних змін квантового стану. Використання фіксованого значення кута $\phi_{ij} = \pi/4$ для всіх наявних ребер дозволяє обмежити глибину квантової схеми та уникнути її надмірного ускладнення, що є важливим в умовах обмежених ресурсів NISQ-пристроїв. Додатково, транспіляція схеми під конкретний квантовий процесор IBM, яка включає мапінг кубітів і оптимізацію вентилів, спрямована на мінімізацію фактичної глибини виконаного кола. Таким чином, запропонований підхід сприяє зменшенню глибини квантового кола порівняно з ітераційними

алгоритмами, такими як QAOA, для яких підвищення якості розв'язку зазвичай пов'язане зі збільшенням глибини схеми (параметра p).

Вплив шумів релаксації та дефазування на стабільність енергетичних очікувань є однією з ключових проблем у NISQ-ері. Алгоритм QASPA-MaxCut демонструє високу стабільність результатів навіть при виконанні на реальних квантових пристроях, що піддаються впливу шуму. Це підтверджується порівняльним аналізом локального та хмарного виконання, наведеним вище, де QASPA коректно ідентифікував оптимальні розв'язки для тестового графа, незважаючи на наявність фізичних шумів. Стійкість алгоритму QASPA до шуму може бути пояснена кількома факторами. По-перше, одноітераційний характер алгоритму зменшує загальний час виконання квантової схеми, що, у свою чергу, обмежує вплив декогеренції. По-друге, у межах дослідження застосовано підхід до врахування апаратних обмежень NISQ-пристроїв, що включає оптимізацію топології квантових схем і використання методів пом'якшення похибок, таких як measurement error mitigation, zero-noise extrapolation та dynamical decoupling. Зазначені методи дозволяють оцінити вплив шуму та частково компенсувати його на рівні статистичної обробки результатів, що сприяє підвищенню їх стабільності та точності.

Масштабованість є ключовим критерієм для оцінки потенційної придатності квантових алгоритмів до розв'язання практичних задач. Оцінювання масштабованості алгоритму QASPA проводилося шляхом тестування на повних графах зі збільшенням кількості вершин у діапазоні від 2 до 19. Отримані результати демонструють перевагу QASPA у порівнянні з QAOA в межах досліджуваного діапазону. Алгоритм QASPA забезпечував знаходження оптимальних значень розрізу для всіх розглянутих графів до $N = 19$, тоді як QAOA починав демонструвати субоптимальні результати вже з $N = 10$, із подальшим зниженням якості розв'язків при зростанні розмірності графа. Така поведінка може свідчити про вищу стійкість QASPA до зростання складності задачі, що пов'язано з використанням детермінованої, графо-залежної параметризації квантової схеми. Слід зазначити, що експоненційне

зростання часу виконання при збільшенні кількості кубітів є властивістю класичної симуляції квантових алгоритмів, а не обмеженням самого алгоритму QASPA. Крім того, застосування моделі масштабування кількості запусків (shots), описаної в підрозділі 2.2, забезпечує адаптивний підхід до отримання статистично значущих результатів при зростанні розмірності задачі. Це дозволяє підтримувати стабільну якість розв'язків у досліджуваному діапазоні та створює передумови для подальшого масштабування алгоритму при використанні більш потужних квантових обчислювальних систем.

Таким чином, проведений комплексний аналіз свідчить про те, що алгоритм QASPA відповідає визначеним критеріям оцінки. Зокрема, він демонструє високу якість розв'язків із середнім коефіцієнтом апроксимації, близьким до 1, при цьому мінімальні значення в усіх експериментальних випадках перевищують 0.9. Запропонований підхід також характеризується ефективною організацією обчислювального процесу завдяки одноітераційній моделі, оптимізованою глибиною квантового кола, стійкістю до шумових впливів та здатністю зберігати якість розв'язків при зростанні розмірності задачі. Отримані результати вказують на перспективність алгоритму QASPA для розв'язання задач комбінаторної оптимізації в умовах NISQ-ери та підтверджують його значущість як квантово-гібридного підходу.

Оцінка часової складності алгоритмів є важливим критерієм їхньої практичної придатності, особливо при розв'язанні NP-складних задач, для яких час обчислень зростає експоненційно зі збільшенням розміру вхідних даних. З метою порівняння алгоритму QASPA з QAOA та методом повного перебору було проведено експериментальне вимірювання часу виконання для графів різної розмірності. Отримані результати дозволяють кількісно оцінити ефективність кожного підходу та проаналізувати їхню поведінку при зростанні обчислювального навантаження.

Детальні результати вимірювання часу виконання для графів розміром від 10 до 19 вершин представлені у таблиці 3.2.

Таблиця 3.2

Результати вимірювання часу виконання

Graph Size	QASPA Time (s)	QAOA Time (s)	Brute Force Time (s)
10	0.1007	2.7238	0.0118
11	0.0848	1.9563	0.0281
12	0.0846	3.0849	0.0670
13	0.0852	3.7778	0.1566
14	0.0893	5.6941	0.3497
15	0.0981	6.2593	0.8046
16	0.1144	6.4957	1.8024
17	0.1692	6.0621	4.0995
18	0.2413	7.4381	9.1214
19	0.4643	8.8575	20.0923

На основі отриманих даних та їх візуалізації (рис. 3.15), побудованої за допомогою функції `plot_times()`, наведеної у додатку, можна сформулювати такі висновки щодо порівняльної продуктивності алгоритмів. Крива, що відповідає методу повного перебору (зелена лінія), демонструє виражене експоненційне зростання часу виконання зі збільшенням розміру графа.

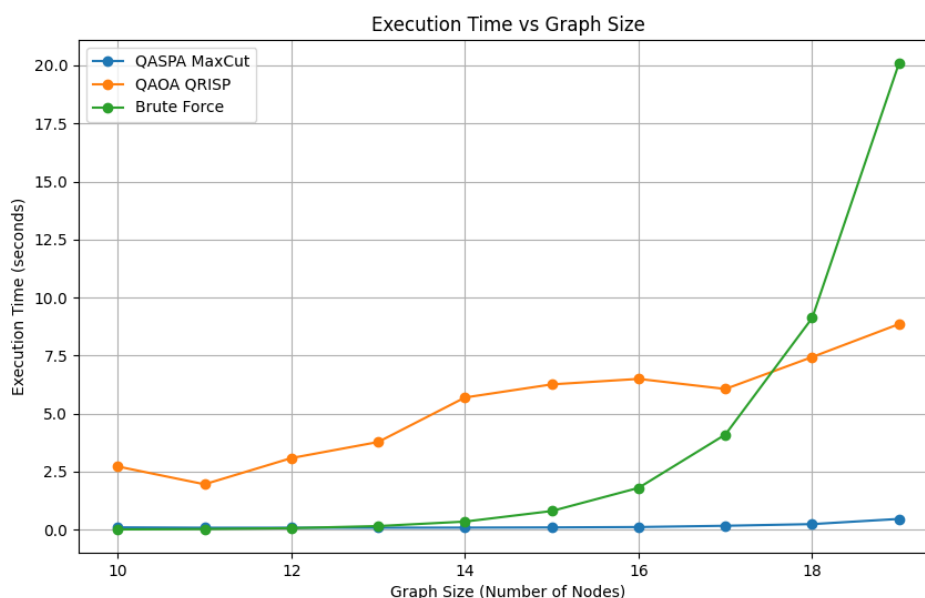


Рис. 3.15. Результати вимірювання часу виконання

Зокрема, для графа з 10 вершинами час виконання становить 0.0118 с, тоді як для 19 вершин він зростає до 20.0923 с. Така динаміка узгоджується з

теоретичною часовою складністю методу повного перебору $O(2^N)$, що обумовлює його практичну непридатність для задач більшої розмірності. У зв'язку з цим, його застосування доцільне переважно як еталонного методу для валідації оптимальності розв'язків на графах малої розмірності.

Час виконання алгоритму QAOA (помаранчева лінія) також демонструє тенденцію до зростання зі збільшенням розміру графа, проте це зростання є менш інтенсивним порівняно з методом повного перебору. Зокрема, для графа з 10 вершинами час виконання становить 2.7238 с, а для 19 вершин – 8.8575 с. На відміну від розглянутих підходів, алгоритм QASPA (синя лінія) характеризується стабільною часовою поведінкою в межах досліджуваного діапазону. Час його виконання для графів розмірності від 10 до 19 вершин варіюється в інтервалі від 0.0846 до 0.4643 с без вираженої тенденції до експоненційного зростання.

Така поведінка зумовлена одноітераційною архітектурою алгоритму, у межах якої параметри квантової схеми визначаються детерміновано на основі структури графа без використання ітераційного класичного оптимізатора. У результаті обчислювальний процес обмежується етапами підготовки графа, формування параметрів, побудови квантової схеми та її одноразового виконання. Відсутність багаторазових квантово-класичних ітерацій дозволяє суттєво зменшити час виконання порівняно з QAOA та методом повного перебору навіть при зростанні розмірності задачі. Отримані результати свідчать про високу часову ефективність алгоритму QASPA та його перспективність для застосування у задачах комбінаторної оптимізації.

Одним із основних напрямів подальших досліджень є розширення масштабованості алгоритму QASPA на графи більшої розмірності. Хоча в межах проведених експериментів було досягнуто рівня 19 кубітів, подальше збільшення кількості доступних кубітів і підвищення їх якості на квантових процесорах відкриває можливість дослідження алгоритму на графах із десятками та сотнями вершин. Це, у свою чергу, вимагатиме додаткової оптимізації як квантової схеми, так і класичних етапів попередньої обробки

даних і постобробки результатів. Іншим важливим напрямом є розширення застосування QASPA на зважені графи. Поточна реалізація орієнтована переважно на незважений варіант задачі Max-Cut, однак теоретична модель, представлена в підрозділі 2.3, передбачає можливість врахування ваг ребер через модифікацію параметрів квантової схеми. Реалізація цього підходу дозволить застосовувати алгоритм до більш складних і практично значущих задач, зокрема у телекомунікаційних мережах із різною пропускнуою здатністю каналів або у фінансових моделях із різною кореляцією активів.

Важливим напрямом подальших досліджень є також аналіз поведінки QASPA на різних топологіях графів. Окрім повних графів, що використовувалися для оцінювання масштабованості, доцільним є дослідження алгоритму на розріджених графах, графах із вираженою кластерною структурою, випадкових графах, а також на реальних мережевих даних (соціальні та транспортні мережі). Такий підхід дозволить виявити особливості функціонування алгоритму в різних структурних умовах і адаптувати його до специфіки прикладних задач.

Подальше вдосконалення квантової схеми QASPA може передбачати дослідження альтернативних підходів до параметризації, а також використання інших типів міксерних операторів, що потенційно дозволить підвищити якість розв'язків або зменшити глибину квантового кола. Інтеграція більш розвинених методів пом'якшення помилок, зокрема варіаційних або нейромережевих підходів, може сприяти підвищенню ефективності алгоритму при виконанні на шумних квантових пристроях.

Перспективним напрямом є також порівняння QASPA з іншими сучасними квантовими алгоритмами для задачі Max-Cut, окрім базових реалізацій QAOA. До таких підходів можна віднести квантовий відпал, глибші варіанти QAOA, а також інші гібридні варіаційні алгоритми. Це дозволить більш точно визначити місце QASPA у загальній архітектурі квантових оптимізаційних методів.

З огляду на універсальність задачі Max-Cut, значний потенціал має адаптація QASPA до розв'язання інших NP-складних задач, які можуть бути зведені до відповідної формалізації, зокрема через модель Ізінга. Це відкриває можливості застосування алгоритму у широкому спектрі прикладних галузей, включаючи логістику, квантову хімію, матеріалознавство та фармакологію, де задачі оптимізації відіграють ключову роль.

Таким чином, алгоритм QASPA можна розглядати як вагомий крок у напрямку практичного застосування квантових обчислень для розв'язання задач комбінаторної оптимізації. Запропонований підхід поєднує високу якість розв'язків, ефективну організацію обчислювального процесу, стійкість до шумових впливів та здатність до масштабування в межах дослідженого діапазону задач. Отримані результати свідчать про перспективність алгоритму QASPA як інструменту для подальших досліджень і розвитку квантово-гібридних методів оптимізації.

Висновки до розділу 3

Обґрунтовано вибір платформи Qiskit та реалізовано запропонований квантово-гібридний алгоритм QASPA, виконання якого здійснено локально за допомогою симулятора AerSimulator з різною кількістю запусків. Проведено перетворення теоретичних моделей у функціональний програмний прототип та його валідацію на квантових симуляторах та реальному апаратному забезпеченні IBM Quantum.

Проведено емпіричне дослідження продуктивності, стійкості та масштабованості алгоритму в умовах сучасної NISQ-ери. Програмно реалізовано алгоритм QASPA, що включає класичну підготовку графа, формування матриці суміжності та генерацію матриці кутів із фіксованим значенням $\pi/4$ для наявних ребер. Встановлено, що така параметризація забезпечує інтерференційне підсилення оптимальних конфігурацій та обмежує глибину квантової схеми. Квантову схему побудовано із застосуванням шару гейтів Адамара та композиції CX-Ry-CX для кожного

ребра графа, що дозволяє ефективно відобразити топологію задачі у квантовому представленні.

Проведено запуск алгоритму QASPA на реальних квантових комп'ютерах IBM Heron R1 та Heron R2. Виявлено, що в умовах впливу шуму та декогеренції алгоритм зберігає здатність до ідентифікації оптимальних розв'язків. Встановлено, що оптимальні бітові конфігурації (зокрема, 01010 для тестового графа) мають найбільшу ймовірність навіть за наявності шуму, що підтверджує робастність алгоритму. Зафіксовано необхідність урахування апаратної сумісності, зокрема обмежень транспіляції для різних квантових архітектур.

Проведено комплексне порівняння із методом повного перебору та алгоритмом QAOA, реалізованим у фреймворку Qrisp. Дослідження виконано на двох вибірках: 17 довільних графів (3-7 вершин) та 65 графів різних типів (циклічні, повні, випадкові) у діапазоні 3-14 вершин.

За критерієм якості розв'язку встановлено, що алгоритм QASPA забезпечує досягнення оптимального або квазіоптимального значення функції вартості для всіх 82 тестових випадків. Виявлено, що середній коефіцієнт апроксимації для QASPA знаходиться у діапазоні 0.98-1.00, а медіанне значення дорівнює 1.0, що свідчить про концентрацію результатів навколо глобального оптимуму. Встановлено, що значення коефіцієнта апроксимації у всіх випадках перевищує порогове значення $p > 0.9$, визначене відносно еталонного розв'язку, отриманого методом повного перебору.

Досліджено показник успішності (success rate) та встановлено, що для алгоритму QASPA він становить 100% для всіх досліджених графів, незалежно від їх типу та розмірності. Виявлено відсутність деградації якості розв'язків при переході від регулярних до стохастичних структур графів.

Для алгоритму QAOA встановлено, що значення коефіцієнта апроксимації знаходяться у діапазоні 0.88-0.96 залежно від типу графа, причому для щільних графів (повні та Erdős–Rényi з $p \geq 0.5$) спостерігається зниження до рівня 0.63-0.90. Виявлено, що показник success rate для QAOA

зменшується зі збільшенням розмірності задачі та щільності графа і може знижуватися до рівня 50-60%, що свідчить про нестабільність результатів та залежність від топології графа.

Встановлено, що запропонований алгоритм QASPA не потребує ітераційної класичної оптимізації параметрів, що дозволяє уникнути проблем локальних мінімумів та плато градієнта, характерних для варіаційних квантових алгоритмів. Виявлено, що детермінована параметризація забезпечує стабільність результатів та відтворюваність експериментів.

Досліджено часові характеристики алгоритмів та встановлено, що метод повного перебору демонструє експоненційне зростання часу виконання від 0.0118 секунд ($n=10$) до 20.0923 секунд ($n=19$). Для алгоритму QAOA виявлено зростання часу виконання від 2.7238 секунд до 8.8575 секунд у тому ж діапазоні. На відміну від цього, алгоритм QASPA демонструє стабільний час виконання в межах 0.0846-0.4643 секунд, що підтверджує його високу ефективність та масштабованість.

Встановлено, що алгоритм QASPA перевершує QAOA за показниками коефіцієнта апроксимації, стабільності отриманих розв'язків та масштабованості. Виявлено, що застосування детермінованої параметризації квантової схеми забезпечує досягнення оптимальних або квазіоптимальних рішень із коефіцієнтом апроксимації 0.98-1.00 та success rate на рівні 100% незалежно від типу графа. Отримані результати підтверджують ефективність запропонованого підходу для розв'язання NP-складних задач комбінаторної оптимізації в умовах обмежених ресурсів квантових обчислювальних систем.

Матеріали розділу 3 подано в таких публікаціях автора [2, 5].

ЗАГАЛЬНІ ВИСНОВКИ

Відповідно до поставленої мети та завдань дослідження з розробки методів і алгоритмів квантово-гібридного розв'язання задачі максимального розрізу, отримано такі **основні результати**: проаналізовано розвиток квантових технологій та сучасний стан класичних, квантових і квантово-гібридних підходів до розв'язання задач комбінаторної оптимізації; розроблено квантово-гібридний алгоритм QASPA для розв'язання задачі Max-Cut на основі детермінованої графо-залежної параметризації квантової схеми; розроблено метод масштабування кількості квантових запусків для підвищення статистичної надійності ідентифікації оптимального або квазіоптимального розв'язку; виконано програмну реалізацію алгоритму QASPA із використанням платформи Qiskit та проведено його експериментальну валідацію на квантових симуляторах і реальному апаратному забезпеченні IBM Quantum; проведено порівняльний аналіз QASPA з алгоритмом QAOA та методом повного перебору за критеріями якості розв'язку, стабільності, масштабованості та часу виконання.

Результати дослідження дали підстави для таких **висновків**:

1. Проведений комплексний аналіз розвитку квантових технологій та сучасного стану методів розв'язання задач комбінаторної оптимізації дозволив встановити перспективність використання квантових і квантово-гібридних підходів для задачі максимального розрізу. Визначено, що Max-Cut є фундаментальною NP-складною задачею, важливою для аналізу мереж, кластеризації, маршрутизації, планування ресурсів та оптимізації графових структур. Точні класичні методи, зокрема метод повного перебору, забезпечують знаходження оптимального розв'язку, однак характеризуються експоненційним зростанням кількості можливих конфігурацій при збільшенні кількості вершин графа. Апроксимаційні та евристичні підходи зменшують обчислювальні витрати, проте не завжди забезпечують стабільну якість результатів. Виявлено обмеження квантово-гібридних алгоритмів, зокрема QAOA, пов'язані зі складністю параметричного простору, залежністю від

початкових умов, потребою в багаторазовому виконанні квантової схеми та чутливістю до шумів, що зумовило необхідність розроблення підходу з детермінованим визначенням параметрів квантової схеми.

2. Розроблено квантово-гібридний алгоритм QASPA для розв'язання задачі Max-Cut, який базується на детермінованій графо-залежній параметризації квантової схеми без використання багатоітераційної класичної оптимізації. Запропоновано метод перетворення графа у квантове представлення на основі матриці суміжності та матриці кутів, у якій для наявних ребер використовується фіксоване значення $\pi/4$, а для відсутніх ребер – нульове значення. Це забезпечує формальну відповідність між структурою графа та параметрами квантової схеми. Розроблено метод формування квантової схеми, що включає початковий шар гейтів Адамара для підготовки рівномірної суперпозиції можливих бітових конфігурацій і подальше застосування композиції CX-Ry-CX для ребер графа. Встановлено, що одноітераційна архітектура QASPA дозволяє зменшити кількість класично-квантових обмінів, скоротити загальний час виконання алгоритму та усунути залежність від класичного оптимізаційного циклу.

3. Розроблено метод масштабування кількості квантових запусків на основі афінно-експоненційної апроксимації, який дозволяє визначати статистично достатню кількість вимірювань для ідентифікації оптимального або квазіоптимального розв'язку при зростанні розмірності графа. Метод враховує збільшення простору можливих бітових конфігурацій зі зростанням кількості вершин і дає змогу формалізувати вибір кількості запусків до початку виконання алгоритму. Визначено, що попереднє встановлення кількості квантових запусків з урахуванням розмірності задачі зменшує ризик втрати оптимального розв'язку через недостатній обсяг статистики та запобігає необґрунтованому збільшенню обчислювальних витрат. Поєднання графо-залежної параметризації, одноітераційної структури виконання та попереднього визначення кількості запусків підвищує відтворюваність результатів і забезпечує більш обґрунтоване проведення експериментів.

4. Виконано програмну реалізацію алгоритму QASPA із використанням платформи Qiskit та проведено його експериментальну валідацію на квантових симуляторах і реальному апаратному забезпеченні IBM Quantum. Розроблена реалізація охоплює повний цикл роботи алгоритму: формування вхідного графа, побудову матриці суміжності, обчислення матриці кутів, генерацію квантової схеми, виконання вимірювань, обробку отриманих бітових рядків і визначення значення цільової функції Max-Cut. Проведено комплексне експериментальне дослідження на 82 тестових графах різних типів, що дозволило оцінити поведінку алгоритму для графів різної структури, щільності та розмірності. У результаті встановлено, що алгоритм QASPA забезпечує досягнення оптимальних або квазіоптимальних розв'язків із середнім коефіцієнтом апроксимації 0.98-1.00 та 100% рівнем успішності незалежно від типу графа в межах проведених експериментів. Отримані результати на симуляторах і реальному квантовому апаратному забезпеченні підтверджують коректність запропонованого підходу та можливість його подальшого використання для дослідження задач комбінаторної оптимізації.

5. Проведено порівняльний аналіз алгоритму QASPA з квантово-гібридним алгоритмом QAOA та методом повного перебору за критеріями якості розв'язку, стабільності, масштабованості та часу виконання. У результаті порівняння встановлено, що QASPA забезпечує стабільну якість розв'язків, тоді як метод повного перебору може використовуватися як еталонний інструмент лише для малих графів, але втрачає практичну ефективність при збільшенні кількості вершин. Показано, що на відміну від QASPA, алгоритм QAOA демонструє зниження якості розв'язків при зростанні розмірності задачі: коефіцієнт апроксимації для QAOA перебуває в межах 0.63-0.96, а показник успішності може знижуватися до 50-60%. Установлено, що алгоритм QASPA характеризується стабільним часом виконання в межах 0.0846-0.4643 с для графів із 10-19 вершинами без експоненційного зростання, характерного для повного перебору. Показано, що детермінована параметризація QASPA дозволяє уникнути проблем локальних

мінімумів і плато градієнта, забезпечуючи стабільність, відтворюваність і передбачуваність результатів.

Практичне значення отриманих результатів полягає в тому, що запропонований алгоритм QASPA, методи його параметризації та програмна реалізація можуть бути використані для дослідження задач комбінаторної оптимізації на квантових симуляторах і хмарних квантових платформах. Результати роботи можуть застосовуватися для побудови експериментальних квантово-гібридних програмних засобів, аналізу графових структур, тестування квантових алгоритмів на пристроях епохи NISQ, а також для подальшого розвитку навчальних і дослідницьких програм у галузі квантового програмування.

Виконане дослідження не вичерпує всіх аспектів поставленої проблеми. Продовження наукових досліджень за даною проблематикою доцільно у таких **напрямах**:

- проведення додаткових експериментів із графами більшої розмірності та складнішої структури, зокрема зваженими, розрідженими, щільними, регулярними, планарними та випадковими графами різних моделей;
- дослідження можливостей масштабування алгоритму QASPA для графів із більшою кількістю вершин шляхом декомпозиції великих графів на підграфи та подальшого узгодження часткових розв'язків;
- адаптація запропонованого підходу до інших задач комбінаторної оптимізації графової природи, зокрема Max-k-Cut, задачі пошуку максимального незалежного набору, розфарбування графа, маршрутизації та кластеризації;
- удосконалення правил формування матриці кутів із урахуванням ваг ребер, степенів вершин, локальної щільності графа, спектральних характеристик матриці суміжності та інших структурних ознак.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Голубничий Д., Коломійцев О., Третяк В., Калачова В., Закіров З., Полтавський Е., Кудряшов В., Рибальченко А., Любченко О. Інформаційна технологія використання гарантованих прогнозів під час рішення задач комбінаторної оптимізації. *Grail of Science*. 2023. № 24. С. 302–312. DOI: 10.36074/grail-of-science.17.02.2023.056.
2. Сапожник Д. О. Процес перетворення високорівневих квантових вентилів у базові для виконання на реальному квантовому комп'ютері. *Вісник Херсонського національного технічного університету*. 2025. Т. 2, № 2(93). С. 327–332. DOI: 10.35546/kntu2078-4481.2025.2.2.40.
3. Сапожник Д. О., Морозов А. В. Економічні та промислові застосування квантових комп'ютерів. *Сучасні інформаційні системи та технології: матеріали VII Всеукр. наук. практ. інтернет-конф. за тематикою «Сучасні комп'ютерні системи та мережі в управлінні»* (29 листопада 2024 р., м. Херсон, м. Хмельницький) / за ред. А. А. Григорової. Херсон: Книжкове видавництво ФОП Вишемирський В. 2024. С. 113-116.
4. Сапожник Д. О. Аналіз можливостей мови програмування Q# шляхом реалізації програми для генерації випадкових чисел. *Технічна інженерія*. 2024. № 1(93). С. 240–245. DOI: 10.26642/ten-2024-1(93)-240-245.
5. Сапожник Д. О. Розв'язання задачі максимального розрізу графа за допомогою квантово-гібридного амплітудно-стохастичного алгоритму. *Національний технічний університет “Харківський політехнічний інститут.”* 2025. Т. 2, № 2 (14). С. 96–101. DOI: 10.20998/2079-0023.2025.02.13.
6. Aasen D. et al. Blueprint for fault-tolerant quantum computation with topological qubit arrays. *Physical Review Research*. 2025. Vol. 7, Issue 4. P. 041002. DOI: 10.1103/qx36-4rv1.
7. AbuGhanem M. IBM quantum computers: evolution, performance, and future directions. *The Journal of Supercomputing*. 2025. Vol. 81, Issue 5. P. 687. DOI: 10.1007/s11227-025-07047-7.

8. AbuGhanem M., Eleuch H. Full quantum tomography study of Google's Sycamore gate on IBM's quantum computers. *EPJ Quantum Technology*. 2024. Vol. 11, Issue 1. P. 36. DOI: 10.1140/epjqt/s40507-024-00248-8.
9. Adubi S. A., Oladipupo O. O., Olugbara O. O. Evolutionary Algorithm-Based Iterated Local Search Hyper-Heuristic for Combinatorial Optimization Problems. *Algorithms*. 2022. Vol. 15, Issue 11. P. 405. DOI: 10.3390/a15110405.
10. Benito C., López E., Peropadre B., Bermudez A. Comparative study of quantum error correction strategies for the heavy-hexagonal lattice. *Quantum*. 2025. Vol. 9. P. 1623. DOI: 10.22331/q-2025-02-06-1623.
11. Caragiannis I., Jiang Z. Computing Better Approximate Pure Nash Equilibria in Cut Games via Semidefinite Programming. *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*. 2023. P. 710–722. DOI: 10.1145/3564246.3585236.
12. Cheng B., Deng X.-H., Gu X., He Y., Hu G., Huang P., Li J., Lin B.-C., Lu D., Lu Y., Qiu C., Wang H., Xin T., Yu S., Yung M.-H., Zeng J., Zhang S., Zhong Y., Peng X., Nori F., Yu D. Noisy intermediate-scale quantum computers. *Frontiers of Physics*. 2023. Vol. 18, Issue 2. P. 21308. DOI: 10.1007/s11467-022-1249-z.
13. Cuéllar M. P. What we can do with one qubit in quantum machine learning: ten classical machine learning problems that can be solved with a single qubit. *Quantum Machine Intelligence*. 2024. Vol. 6, Issue 2. P. 76. DOI: 10.1007/s42484-024-00210-y.
14. De Jesus G. F., Teixeira E. S., Galvão L. Q., Da Silva M. H. F., Neto M. Q. N., Fernandez B. O., De Lima A. M., De Aguiar Silva E. D. R., Cruz C. D. S. Evaluating Variational Quantum Eigensolver Approaches for Simplified Models of Molecular Systems: A Case Study on Protocatechuic Acid. *Molecules*. 2024. Vol. 30, Issue 1. P. 119. DOI: 10.3390/molecules30010119.
15. Evans P. W., Hangleiter D., Thébault K. P. Y. How to Engineer a Quantum Wavefunction. *Philosophy of Science*. 2024. Vol. 91, Issue 1. P. 37–57. DOI: 10.1017/psa.2023.87.

16. Fedorov D. A., Alexeev Y., Gray S. K., Otten M. Unitary Selective Coupled-Cluster Method. *Quantum*. 2022. Vol. 6. P. 703. DOI: 10.22331/q-2022-05-02-703.
17. Feng G., Lu D., Li J., Xin T., Zeng B. Quantum Computing: Principles and Applications. *SPIN*. 2023. Vol. 13, Issue 03. P. 2330001. DOI: 10.1142/S2010324723300013.
18. Feniou C., Hassan M., Traoré D., Giner E., Maday Y., Piquemal J.-P. Overlap-ADAPT-VQE: practical quantum chemistry on quantum computers via overlap-guided compact Ansätze. *Communications Physics*. 2023. Vol. 6, Issue 1. P. 192. DOI: 10.1038/s42005-023-01312-y.
19. Gemeinhardt F., Garmendia A., Wimmer M., Wille R. A Model-Driven Framework for Composition-Based Quantum Circuit Design. *ACM Transactions on Quantum Computing*. 2024. Vol. 5, Issue 4. P. 1–36. DOI: 10.1145/3688856.
20. Gevorgyan H. L., Vitanov N. V. Ultrahigh-fidelity composite quantum phase gates. *Physical Review A*. 2024. Vol. 109, Issue 5. P. 052625. DOI: 10.1103/PhysRevA.109.052625.
21. Gonzalez-Ayala P., Alejo-Reyes A., Cuevas E., Mendoza A. A Modified Simulated Annealing (MSA) Algorithm to Solve the Supplier Selection and Order Quantity Allocation Problem with Non-Linear Freight Rates. *Axioms*. 2023. Vol. 12, Issue 5. P. 459. DOI: 10.3390/axioms12050459.
22. Herbert S. Quantum computing for data-centric engineering and science. *Data-Centric Engineering*. 2022. Vol. 3. P. e36. DOI: 10.1017/dce.2022.36.
23. Jansson C. An Alternative Approach to Probability in Quantum Information Science. *Y: Quantum Information Science - Recent Advances and Computational Science Applications* ed. R. Steijl. IntechOpen, 2024. URL: <https://www.intechopen.com/chapters/1176955> (accessed 9.April.2025).
24. Jing H., Wang Y., Li Y. Data-driven quantum approximate optimization algorithm for power systems. *Communications Engineering*. 2023. Vol. 2, Issue 1. P. 12. DOI: 10.1038/s44172-023-00061-8.

25. Keppens J., Eggerickx Q., Levajac V., Simion G., Sorée B. Qudit vs. qubit: Simulated performance of error-correction codes in higher dimensions. *Physical Review A*. 2025. Vol. 112, Issue 3. P. 032435. DOI: 10.1103/2w52-qd2j.
26. Kungurtsev V., Korpas G., Marecek J., Zhu E. Y. Iteration Complexity of Variational Quantum Algorithms. *Quantum*. 2024. Vol. 8. P. 1495. DOI: 10.22331/q-2024-10-10-1495.
27. Lancaster J. L., Allen D. B. Simulating spin dynamics with quantum computers. *American Journal of Physics*. 2025. Vol. 93, Issue 1. P. 98–109. DOI: 10.1119/5.0112717.
28. Li H., Lu K., Han Z., Qin H., Ju M., Liu S. Research on Qubit Mapping Technique Based on Batch SWAP Optimization. *International Journal of Advanced Computer Science and Applications*. 2023. Vol. 14, Issue 12. DOI: 10.14569/IJACSA.2023.0141217.
29. Lubinski T., Coffrin C., McGeoch C., Sathe P., Apanavicius J., Bernal Neira D., Quantum Economic Development Consortium(QED-C) Collaboration. Optimization Applications as Quantum Performance Benchmarks. *ACM Transactions on Quantum Computing*. 2024. Vol. 5, Issue 3. P. 1–44. DOI: 10.1145/3678184.
30. Maring N., Fyrrillas A., Pont M., Ivanov E., Stepanov P., Margaria N., Hease W., Pishchagin A., Lemaître A., Sagnes I., Au T. H., Boissier S., Bertasi E., Baert A., Valdivia M., Billard M., Acar O., Brioussel A., Mezher R., Wein S. C., Salavrakos A., Sinnott P., Fioretto D. A., Emeriau P.-E., Belabas N., Mansfield S., Senellart P., Senellart J., Somaschi N. A versatile single-photon-based quantum computing platform. *Nature Photonics*. 2024. Vol. 18, Issue 6. P. 603–609. DOI: 10.1038/s41566-024-01403-4.
31. Oliva Mary Fernandes, Nisha, Omkar Naik, Pavan Kumar, Dr. Pradeep V. Quantum Computers and their Application in Future Era. *International Journal of Advanced Research in Science, Communication and Technology*. 2024. P. 227–230. DOI: 10.48175/IJARSCT-19331.

32. Perelshtein M. R., Pakhomchik A. I., Melnikov Ar. A., Podobrii M., Termanova A., Kreidich I., Nuriev B., Iudin S., Mansell C. W., Vinokur V. M. NISQ-compatible approximate quantum algorithm for unconstrained and constrained discrete optimization. *Quantum*. 2023. Vol. 7. P. 1186. DOI: 10.22331/q-2023-11-21-1186.

33. Ponce M., Herrman R., Lotshaw P. C., Powers S., Siopsis G., Humble T., Ostrowski J. Graph decomposition techniques for solving combinatorial optimization problems with variational quantum algorithms. *Quantum Information Processing*. 2025. Vol. 24, Issue 2. P. 60. DOI: 10.1007/s11128-025-04675-z.

34. Qiang X., Ma S., Song H. Quantum Walk Computing: Theory, Implementation, and Application. *Intelligent Computing*. 2024. Vol. 3. P. 0097. DOI: 10.34133/icomputing.0097.

35. Rakyta P., Morse G., Náadori J., Majnay-Takács Z., Mencer O., Zimborás Z. Highly optimized quantum circuits synthesized via data-flow engines. *Journal of Computational Physics*. 2024. Vol. 500. P. 112756. DOI: 10.1016/j.jcp.2024.112756.

36. Rosa E. C. R., Duzzioni E. I., De Santiago R. Optimizing Gate Decomposition for High-Level Quantum Programming. *Quantum*. 2025. Vol. 9. P. 1659. DOI: 10.22331/q-2025-03-12-1659.

37. Roy T., Jiang L., Schuster D. I. Deterministic Grover search with a restricted oracle. *Physical Review Research*. 2022. Vol. 4, Issue 2. P. L022013. DOI: 10.1103/PhysRevResearch.4.L022013.

38. Ryan-Anderson C., Bohnet J. G., Lee K., Gresh D., Hankin A., Gaebler J. P., Francois D., Chernoguzov A., Lucchetti D., Brown N. C., Gatterman T. M., Halit S. K., Gilmore K., Gerber J. A., Neyenhuis B., Hayes D., Stutz R. P. Realization of Real-Time Fault-Tolerant Quantum Error Correction. *Physical Review X*. 2021. Vol. 11, Issue 4. P. 041058. DOI: 10.1103/PhysRevX.11.041058.

39. Sakuler W., Oberreuter J. M., Aiolfi R., Asproni L., Roman B., Schiefer J. A real-world test of portfolio optimization with quantum annealing. *Quantum*

Machine Intelligence. 2025. Vol. 7, Issue 1. P. 43. DOI: 10.1007/s42484-025-00268-2.

40. Sureshababu S. H., Herman D., Shaydulin R., Basso J., Chakrabarti S., Sun Y., Pistoia M. Parameter Setting in Quantum Approximate Optimization of Weighted Problems. *Quantum*. 2024. Vol. 8. P. 1231. DOI: 10.22331/q-2024-01-18-1231.

41. Tomar V., Bansal M., Singh P. Metaheuristic Algorithms for Optimization: A Brief Review. *RAiSE-2023*. 2024. P. 238. DOI: 10.3390/engproc2023059238.

42. Van Dam W., Mykhailova M., Soeken M. Using Azure Quantum Resource Estimator for Assessing Performance of Fault Tolerant Quantum Computation. *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*. 2023. P. 1414–1419. DOI: 10.1145/3624062.3624211.

43. Watts A. B., Chowdhury A., Epperly A., Helton J. W., Klep I. Relaxations and Exact Solutions to Quantum Max Cut via the Algebraic Structure of Swap Operators. *Quantum*. 2024. Vol. 8. P. 1352. DOI: 10.22331/q-2024-05-22-1352.

44. Wei K. X., Lauer I., Pritchett E., Shanks W., McKay D. C., Javadi-Abhari A. Native Two-Qubit Gates in Fixed-Coupling, Fixed-Frequency Transmons Beyond Cross-Resonance Interaction. *PRX Quantum*. 2024. Vol. 5, Issue 2. P. 020338. DOI: 10.1103/PRXQuantum.5.020338.

45. Wei S., Jing B., Zhang X., Liao J., Yuan C., Fan B., Lyu C., Zhou D., Wang Y., Deng G., Song H., Oblak D., Guo G., Zhou Q. Towards Real-World Quantum Networks: A Review. *Laser & Photonics Reviews*. 2022. Vol. 16, Issue 3. P. 2100219. DOI: 10.1002/lpor.202100219.

46. Yazdani A., Von Oppen F., Halperin B. I., Yacoby A. Hunting for Majoranas. *Science*. 2023. Vol. 380, Issue 6651. P. eade0850. DOI: 10.1126/science.ade0850.

47. Zhu X. et al. Combinatorial optimization enhanced by shallow quantum circuits with 104 superconducting qubits. *National Science Review*. 2026. P. nwag124. DOI: 10.1093/nsr/nwag124.
48. Zindorf B., Bose S. Efficient implementation of multicontrolled quantum gates. *Physical Review Applied*. 2025. Vol. 24, Issue 4. P. 044030. DOI: 10.1103/8blx-nfcr.
49. AbuGhanem M. Superconducting quantum computers: who is leading the future? *EPJ Quantum Technology*. 2025. Vol. 12, Issue 1. P. 102. DOI: 10.1140/epjqt/s40507-025-00405-7.
50. Al-Neama M. W., Ahmed I. A., Ali S. M. A parallel algorithm to find the exact solution of the travelling salesman problem. *Indonesian Journal of Electrical Engineering and Computer Science*. 2023. Vol. 31, Issue 2. P. 917. DOI: 10.11591/ijeecs.v31.i2.pp917-924.
51. Arute F. et al. Quantum supremacy using a programmable superconducting processor. *Nature*. 2019. Vol. 574, Issue 7779. P. 505–510. DOI: 10.1038/s41586-019-1666-5.
52. Bartee S. K., Gilbert W., Zuo K., Das K., Tanttu T., Yang C. H., Dumoulin Stuyck N., Pauka S. J., Su R. Y., Lim W. H., Serrano S., Escott C. C., Hudson F. E., Itoh K. M., Laucht A., Dzurak A. S., Reilly D. J. Spin-qubit control with a millikelvin CMOS chip. *Nature*. 2025. Vol. 643, Issue 8071. P. 382–387. DOI: 10.1038/s41586-025-09157-x.
53. Bassman Otfelie L., Urbanek M., Metcalf M., Carter J., Kemper A. F., De Jong W. A. Simulating quantum materials with digital quantum computers. *Quantum Science and Technology*. 2021. Vol. 6, Issue 4. P. 043002. DOI: 10.1088/2058-9565/ac1ca6.
54. Bengio Y., Lodi A., Prouvost A. Machine learning for combinatorial optimization: A methodological tour d’horizon. *European Journal of Operational Research*. 2021. Vol. 290, Issue 2. P. 405–421. DOI: 10.1016/j.ejor.2020.07.063.

55. Berberich J., Fink D. Quantum Computing Through the Lens of Control: A Tutorial Introduction. *IEEE Control Systems*. 2024. Vol. 44, Issue 6. P. 24–49. DOI: 10.1109/MCS.2024.3466448.
56. Blekos K., Brand D., Ceschini A., Chou C.-H., Li R.-H., Pandya K., Summer A. A review on Quantum Approximate Optimization Algorithm and its variants. *Physics Reports*. 2024. Vol. 1068. P. 1–66. DOI: 10.1016/j.physrep.2024.03.002.
57. Blekos K., Brand D., Ceschini A., Chou C.-H., Li R.-H., Pandya K., Summer A. A review on Quantum Approximate Optimization Algorithm and its variants. *Physics Reports*. 2024. Vol. 1068. P. 1–66. DOI: 10.1016/j.physrep.2024.03.002.
58. Blinov S., Wu B., Monroe C. Comparison of cloud-based ion trap and superconducting quantum computer architectures. *AVS Quantum Science*. 2021. Vol. 3, Issue 3. P. 033801. DOI: 10.1116/5.0058187.
59. Borgarino M., Badiali A. Quantum Gates for Electronics Engineers. *Electronics*. 2023. Vol. 12, Issue 22. P. 4664. DOI: 10.3390/electronics12224664.
60. Bouchmal O., Cimoli B., Stabile R., Vegas Olmos J. J., Tafur Monroy I. From classical to quantum machine learning: survey on routing optimization in 6G software defined networking. *Frontiers in Communications and Networks*. 2023. Vol. 4. P. 1220227. DOI: 10.3389/frcmn.2023.1220227.
61. Callison A., Chancellor N. Hybrid quantum-classical algorithms in the noisy intermediate-scale quantum era and beyond. *Physical Review A*. 2022. Vol. 106, Issue 1. P. 010101. DOI: 10.1103/PhysRevA.106.010101.
62. Chan H. H. S., Fitzpatrick N., Segarra-Martí J., Bearpark M. J., Tew D. P. Molecular excited state calculations with adaptive wavefunctions on a quantum eigensolver emulation: reducing circuit depth and separating spin states. *Physical Chemistry Chemical Physics*. 2021. Vol. 23, Issue 46. P. 26438–26450. DOI: 10.1039/D1CP02227J.

63. Chatterjee Y., Bourreau E., Rančić M. J. Solving various NP-hard problems using exponentially fewer qubits on a quantum computer. *Physical Review A*. 2024. Vol. 109, Issue 5. P. 052441. DOI: 10.1103/PhysRevA.109.052441.
64. Chen X. A Comparison of Greedy Algorithm and Dynamic Programming Algorithm. *SHS Web of Conferences*. 2022. Vol. 144. P. 03009. DOI: 10.1051/shsconf/202214403009.
65. Chia H. L. B. Quantum computing and its revolutionary potential. *Advances in Engineering Innovation*. 2023. Vol. 4, Issue 1. P. 26–32. DOI: 10.54254/2977-3903/4/2023022.
66. Chicano F., Luque G., Dahi Z. A., Gil-Merino R. Combinatorial optimization with quantum computers. *Engineering Optimization*. 2025. Vol. 57, Issue 1. P. 208–233. DOI: 10.1080/0305215X.2024.2435538.
67. Clinton L., Bausch J., Cubitt T. Hamiltonian simulation algorithms for near-term quantum hardware. *Nature Communications*. 2021. Vol. 12, Issue 1. P. 4989. DOI: 10.1038/s41467-021-25196-0.
68. Della Giustina D., Londero C., Piazza C., Riccardi B., Romanello R. Quantum encoding of dynamic directed graphs. *Journal of Logical and Algebraic Methods in Programming*. 2024. Vol. 136. P. 100925. DOI: 10.1016/j.jlamp.2023.100925.
69. Eclipse Qrisp. Eclipse Qrisp. URL: <https://www.qrisp.eu/> (accessed 28.February.2026).
70. Egger D. J., Mareček J., Woerner S. Warm-starting quantum optimization. *Quantum*. 2021. Vol. 5. P. 479. DOI: 10.22331/q-2021-06-17-479.
71. Farhi E., Goldstone J., Gutmann S., Zhou L. The Quantum Approximate Optimization Algorithm and the Sherrington-Kirkpatrick Model at Infinite Size. *Quantum*. 2022. Vol. 6. P. 759. DOI: 10.22331/q-2022-07-07-759.
72. Gaspers S., Kim E. J., Ordyniak S., Saurabh S., Szeider S. Don't Be Strict in Local Search! *Proceedings of the AAAI Conference on Artificial Intelligence*. 2021. Vol. 26, Issue 1. P. 486–492. DOI: 10.1609/aaai.v26i1.8128.

73. George H. C. et al. 12-Spin-Qubit Arrays Fabricated on a 300 mm Semiconductor Manufacturing Line. *Nano Letters*. 2025. Vol. 25, Issue 2. P. 793–799. DOI: 10.1021/acs.nanolett.4c05205.

74. Gitonga C. K. The Impact of Quantum Computing on Cryptographic Systems: Urgency of Quantum-Resistant Algorithms and Practical Applications in Cryptography. *European Journal of Information Technologies and Computer Science*. 2025. Vol. 5, Issue 1. P. 1–10. DOI: 10.24018/compute.2025.5.1.146.

75. Google Quantum AI and Collaborators, Acharya R. et al. Quantum error correction below the surface code threshold. *Nature*. 2025. Vol. 638, Issue 8052. P. 920–926. DOI: 10.1038/s41586-024-08449-y.

76. Grabowska A., Gunia A. On quantum computing for artificial superintelligence. *European Journal for Philosophy of Science*. 2024. Vol. 14, Issue 2. DOI: 10.1007/s13194-024-00584-7. URL: <https://doi.org/10.1007/s13194-024-00584-7>.

77. Grange C., Poss M., Bourreau E. An introduction to variational quantum algorithms for combinatorial optimization problems. *4OR*. 2023. Vol. 21, Issue 3. P. 363–403. DOI: 10.1007/s10288-023-00549-1.

78. Guo M. Progress in the Study of the Boolean Satisfiability Problem. *International Journal of Computer Science and Information Technology*. 2024. Vol. 2, Issue 2. P. 217–225. DOI: 10.62051/ijcsit.v2n2.25.

79. Halffmann P., Schäfer L. E., Dächert K., Klamroth K., Ruzika S. Exact algorithms for multiobjective linear optimization problems with integer variables: A state of the art survey. *Journal of Multi-Criteria Decision Analysis*. 2022. Vol. 29, Issue 5–6. P. 341–363. DOI: 10.1002/mcda.1780.

80. He Z., Shaydulin R., Chakrabarti S., Herman D., Li C., Sun Y., Pistoia M. Alignment between initial state and mixer improves QAOA performance for constrained optimization. *npj Quantum Information*. 2023. Vol. 9, Issue 1. P. 121. DOI: 10.1038/s41534-023-00787-5.

81. Hossain M., Rakshit J. K., Chattopadhyay T. Silicon microring resonator based on all-optical reversible modified Fredkin gate: design and analysis. *Optical Engineering*. 2022. Vol. 61, Issue 04. DOI: 10.1117/1.OE.61.4.046111.

82. Houssein E. H., Saeed M. K., Hu G., Al-Sayed M. M. Metaheuristics for Solving Global and Engineering Optimization Problems: Review, Applications, Open Issues and Challenges. *Archives of Computational Methods in Engineering*. 2024. Vol. 31, Issue 8. P. 4485–4519. DOI: 10.1007/s11831-024-10168-6.

83. Huang C., Zhang F., Newman M., Ni X., Ding D., Cai J., Gao X., Wang T., Wu F., Zhang G., Ku H.-S., Tian Z., Wu J., Xu H., Yu H., Yuan B., Szegedy M., Shi Y., Zhao H.-H., Deng C., Chen J. Efficient parallelization of tensor network contraction for simulating quantum computation. *Nature Computational Science*. 2021. Vol. 1, Issue 9. P. 578–587. DOI: 10.1038/s43588-021-00119-7.

84. Ibrahim O. A. S., Younis E. M. G. A Hybrid-Box tool for Explainable and Interactive Simulated Annealing Strategy. 2023. DOI: 10.21203/rs.3.rs-3578275/v1.

85. Ivohin E. V., Gavrylenko V. V., Ivohina K. E. On the recursive algorithm for solving the traveling salesman problem on the basis of the data flow optimization method. *Radio Electronics, Computer Science, Control*. 2023. Issue 3. P. 141. DOI: 10.15588/1607-3274-2023-3-14.

86. Jain N., Coyle B., Kashefi E., Kumar N. Graph neural network initialisation of quantum approximate optimisation. *Quantum*. 2022. Vol. 6. P. 861. DOI: 10.22331/q-2022-11-17-861.

87. Jaradat Y., Alia M., Masoud M., Mansrah A., Jannoud I., Alheyasat O. Roadmap for Simulating Quantum Circuits Utilising IBM's Qiskit Library: Programming Approach. *The Eurasia Proceedings of Science Technology Engineering and Mathematics*. 2023. Vol. 26. P. 624–632. DOI: 10.55549/epstem.1412445.

88. Jaradat Y., Masoud M., Mansrah A., Jannoud I., Alia M. Phases Unlocked: The Crucial Role of Qubit Phases in Quantum Computing. *The Eurasia Proceedings of Science Technology Engineering and Mathematics*. 2024. Vol. 32. P. 406–416. DOI: 10.55549/epstem.1604685.

89. Ji Y., Polian I. Synergistic Dynamical Decoupling and Circuit Design for Enhanced Algorithm Performance on Near-Term Quantum Devices. *Entropy*. 2024. Vol. 26, Issue 7. P. 586. DOI: 10.3390/e26070586.
90. Jiang X., Vandenberghe L. Bregman primal–dual first-order method and application to sparse semidefinite programming. *Computational Optimization and Applications*. 2022. Vol. 81, Issue 1. P. 127–159. DOI: 10.1007/s10589-021-00339-7.
91. Joshi A., Kairali A., Raju R., Athreya A., Vishwakarma S., Ganguly S., P. R. M. Quantum Circuit Optimization of Arithmetic circuits using ZX Calculus. 2023. DOI: 10.21203/rs.3.rs-3052769/v1.
92. Khakhanova A., Chumachenko S., Rakhlis D., Hahanov I., Hahanov V. QUANTUM DIGITAL-ANALOGUE COMPUTING. *Radio Electronics, Computer Science, Control*. 2022. Issue 4. P. 40. DOI: 10.15588/1607-3274-2022-4-4.
93. Khan M. J., Bibi S., Khan M. A., Ali H. S., Khan A. I., Anwar R., Islam F. Investigating Quantum Machine Learning Frameworks and Simulating Quantum Approaches. *The Asian Bulletin of Big Data Management*. 2024. Vol. 4, Issue 4. P. 34–43. DOI: 10.62019/abbdm.v4i4.232.
94. Kiseleva O. M., Prytomanova O. M., Lebediev D. M., Filat O. A. Software for solving the two-stage location-allocation problems. *Problems of applied mathematics and mathematic modeling*. 2023. P. 94–100. DOI: 10.15421/322310.
95. Klingler A., Van Der Eyden M., Stengele S., Reinhart T., De Las Cuevas G. Many bounded versions of undecidable problems are NP-hard. *SciPost Physics*. 2023. Vol. 14, Issue 6. P. 173. DOI: 10.21468/SciPostPhys.14.6.173.
96. Lalit Mohan Trivedi E. Al. Quantum Computing Algorithms for Solving Complex Mathematical Problems. *International Journal on Recent and Innovation Trends in Computing and Communication*. 2023. Vol. 11, Issue 10. P. 2202–2206. DOI: 10.17762/ijritcc.v11i10.8913.
97. Łazuka E., Futa A., Jastrzębska M., Łagód G., Szeląg B., Fatone F. The Applications of Graph Algorithms to Modeling of Integrated Urban Water

Management System. *Advances in Science and Technology Research Journal*. 2022. Vol. 16, Issue 5. P. 277–289. DOI: 10.12913/22998624/155037.

98. Li H., Yang Y., Lv P., Qu J., Wang Z.-H., Sun J., Ying S. Utilizing quantum processor for the analysis of strongly correlated materials. *Physica Scripta*. 2024. Vol. 99, Issue 10. P. 105117. DOI: 10.1088/1402-4896/ad770b.

99. Liu X., Li P., Meng F., Zhou H., Zhong H., Zhou J., Mou L., Song S. Simulated annealing for optimization of graphs and sequences. *Neurocomputing*. 2021. Vol. 465. P. 310–324. DOI: 10.1016/j.neucom.2021.09.003.

100. Maccone L. Schrödinger Cats and Quantum Complementarity. *Foundations of Physics*. 2024. Vol. 54, Issue 1. P. 17. DOI: 10.1007/s10701-023-00750-6.

101. Malimban T. M., Oropesa K. R., Geron C. Z., Comia J. K., Ado R. G., Tubola O. D. Hybrid Approaches for Efficient Simulations of 3-Qubit Quantum Fourier Transform (QFT) Circuit Using Quick Quantum Circuit Simulation (QQCS). *ECTI Transactions on Computer and Information Technology (ECTI-CIT)*. 2024. Vol. 18, Issue 2. P. 185–194. DOI: 10.37936/ecti-cit.2024182.253574.

102. Mohammad Rafeek Khan. Analysis and Implementation of Quantum Gates (X, Z, Y, Hadamard CNOT and CCNOT) by using matrices for Quantum Computing. *Communications on Applied Nonlinear Analysis*. 2024. Vol. 32, Issue 1s. P. 507–520. DOI: 10.52783/cana.v32.2288.

103. Munirah Mohd S., Ilias M Shuhud M., Idrus B., Zainuddin H., Mohamad Jan N., Ridza Wahiddin M., Asyraf Zhahir A. Quantum Computing in The Cloud - A Systematic Literature Review. *International journal of electrical and computer engineering systems*. 2024. Vol. 15, Issue 2. P. 185–200. DOI: 10.32985/ijeces.15.2.7.

104. Neha K., . A. Quantum programming: Working with IBM’S qiskit tool. *The Scientific Temper*. 2023. Vol. 14, Issue 01. DOI: 10.58414/SCIENTIFICTEMPER.2023.14.1.11.

105. Nikolettseas S., Raptopoulos C., Spirakis P. MAX CUT in Weighted Random Intersection Graphs and Discrepancy of Sparse Random Set Systems.

Algorithmica. 2023. Vol. 85, Issue 9. P. 2817–2842. DOI: 10.1007/s00453-023-01121-3.

106. Osaba E., Miranda-Rodriguez P., Oikonomakis A., Petrič M., Ruiz A., Bock S., Kourtis M.-A. Solving Drone Routing Problems with Quantum Computing: A Hybrid Approach Combining Quantum Annealing and Gate-Based Paradigms. 2025 IEEE Congress on Evolutionary Computation (CEC). 2025. P. 1–8. DOI: 10.1109/CEC65147.2025.11042978.

107. Patti T. L., Kossaifi J., Anandkumar A., Yelin S. F. Quantum Goemans-Williamson Algorithm with the Hadamard Test and Approximate Amplitude Constraints. Quantum. 2023. Vol. 7. P. 1057. DOI: 10.22331/q-2023-07-12-1057.

108. Pérez-Castillo R., Serrano M. A., Piattini M. Software modernization to embrace quantum technology. Advances in Engineering Software. 2021. Vol. 151. P. 102933. DOI: 10.1016/j.advengsoft.2020.102933.

109. Petropoulos F. et al. Operational Research: methods and applications. Journal of the Operational Research Society. 2024. Vol. 75, Issue 3. P. 423–617. DOI: 10.1080/01605682.2023.2253852.

110. Practice, Q-CTRL. Practice | Q-CTRL. URL: <https://black.qctrl.com/practice/single-qubit> (accessed 21.January.2026).

111. Prántare F. Dividing the Indivisible : Algorithms, Empirical Advances, and Complexity Results for Value-Maximizing Combinatorial Assignment Problems. 2024. Vol. 2382. DOI: 10.3384/9789180756013.

112. Pyrih Ya., Klymash M., Pyrih Yu., Lavriv O. Genetic algorithm as a tool for solving optimisation problems. Information and communication technologies, electronic engineering. 2023. Vol. 3, Issue 2. P. 95–107. DOI: 10.23939/ictee2023.02.095.

113. quantum. quantum - Search | ScienceDirect.com. URL: <https://www.sciencedirect.com/search?q=quantum&subjectAreas=1700&sortBy=date> (accessed 20.January.2026).

114. Quantum Technology and Application Consortium. Industry quantum computing applications. EPJ Quantum Technology. 2021. Vol. 8, Issue 1. P. 25. DOI: 10.1140/epjqt/s40507-021-00114-x.

115. Raviolo D., Civera M., Zanotti Fragonara L. A Comparative Analysis of Optimization Algorithms for Finite Element Model Updating on Numerical and Experimental Benchmarks. Buildings. 2023. Vol. 13, Issue 12. P. 3010. DOI: 10.3390/buildings13123010.

116. Sapozhnyk D. Quantum Approximate Optimization Algorithm for the Max-Cut Problem : JavaScript Programming Language Implementation. 2024. P. 1 Online-Ressource (8 Seiten). DOI: 10.25673/118110.

117. Sapozhnyk D. O. Methods and areas of application of quantum computers in cryptography and logistics. Scientific notes of Taurida National V.I. Vernadsky University. Series: Technical Sciences. 2025. Vol. 2, Issue 5. P. 260–265. DOI: 10.32782/2663-5941/2025.5.2/34.

118. Schaller D., Geiß M., Hellmuth M., Stadler P. F. Heuristic algorithms for best match graph editing. Algorithms for Molecular Biology. 2021. Vol. 16, Issue 1. P. 19. DOI: 10.1186/s13015-021-00196-3.

119. Şeker O., Ekim T., Taşkın Z. C. An exact cutting plane algorithm to solve the selective graph coloring problem in perfect graphs. European Journal of Operational Research. 2021. Vol. 291, Issue 1. P. 67–83. DOI: 10.1016/j.ejor.2020.09.017.

120. Shaydulin R. et al. Evidence of scaling advantage for the quantum approximate optimization algorithm on a classically intractable problem. Science Advances. 2024. Vol. 10, Issue 22. P. eadm6761. DOI: 10.1126/sciadv.adm6761.

121. Shokri S., Rafibakhsh S., Pooshgan R., Faeghi R. Implementation of a quantum algorithm to estimate the energy of a particle in a finite square well potential on IBM quantum computer. The European Physical Journal Plus. 2021. Vol. 136, Issue 7. P. 762. DOI: 10.1140/epjp/s13360-021-01743-y.

122. Sidak Bawa. Exploring Quantum Computing: Principles and Applications. *Journal of Quantum Science and Technology*. 2024. Vol. 1, Issue 3. P. 57–69. DOI: 10.36676/jqst.v1.i3.27.
123. Silva R., Schouery R. A Branch-and-Cut-and-Price Algorithm for Cutting Stock and Related Problems. *Revista Eletrônica de Iniciação Científica em Computação*. 2024. Vol. 22, Issue 1. P. 31–40. DOI: 10.5753/reic.2024.4646.
124. Skitsko V. I., Voinikov M. Yu. Application of Evolutionary Algorithms for Delivery Route Optimization under Wartime Conditions. *Business Inform.* 2024. Vol. 11, Issue 562. P. 323–332. DOI: 10.32983/2222-4459-2024-11-323-332.
125. Steinacker P. et al. Industry-compatible silicon spin-qubit unit cells exceeding 99% fidelity. *Nature*. 2025. Vol. 646, Issue 8083. P. 81–87. DOI: 10.1038/s41586-025-09531-9.
126. Streif M., Leib M., Wudarski F., Rieffel E., Wang Z. Quantum algorithms with local particle-number conservation: Noise effects and error correction. *Physical Review A*. 2021. Vol. 103, Issue 4. P. 042412. DOI: 10.1103/PhysRevA.103.042412.
127. Tate R., Moondra J., Gard B., Mohler G., Gupta S. Warm-Started QAOA with Custom Mixers Provably Converges and Computationally Beats Goemans-Williamson's Max-Cut at Low Circuit Depths. *Quantum*. 2023. Vol. 7. P. 1121. DOI: 10.22331/q-2023-09-26-1121.
128. The Quantum Insider. The Quantum Insider. URL: <https://app.thequantuminsider.com/> (accessed 20.January.2026).
129. Tilly J., Chen H., Cao S., Picozzi D., Setia K., Li Y., Grant E., Wossnig L., Rungger I., Booth G. H., Tennyson J. The Variational Quantum Eigensolver: A review of methods and best practices. *Physics Reports*. 2022. Vol. 986. P. 1–128. DOI: 10.1016/j.physrep.2022.08.003.
130. Truger F., Barzen J., Bechtold M., Beisel M., Leymann F., Mandl A., Yussupov V. Warm-Starting and Quantum Computing: A Systematic Mapping Study. *ACM Computing Surveys*. 2024. Vol. 56, Issue 9. P. 1–31. DOI: 10.1145/3652510.

131. Vijendran V., Das A., Koh D. E., Assad S. M., Lam P. K. An expressive ansatz for low-depth quantum approximate optimisation. *Quantum Science and Technology*. 2024. Vol. 9, Issue 2. P. 025010. DOI: 10.1088/2058-9565/ad200a.

132. Watanabe H. C., Raymond R., Ohnishi Y.-Y., Kaminishi E., Sugawara M. Optimizing Parameterized Quantum Circuits With Free-Axis Single-Qubit Gates. *IEEE Transactions on Quantum Engineering*. 2023. Vol. 4. P. 1–16. DOI: 10.1109/TQE.2023.3286411.

133. Weon W. Understanding Basic Principles of Qubits in Quantum Computing. *International Journal for Research in Applied Science and Engineering Technology*. 2025. Vol. 13, Issue 2. P. 1490–1499. DOI: 10.22214/ijraset.2025.67121.

134. Yue B., Xue S., Pan Y., Jiang M. Quantum approximate optimization algorithm in non-Markovian quantum systems. *Physica Scripta*. 2023. Vol. 98, Issue 10. P. 105104. DOI: 10.1088/1402-4896/acf6e8.

135. Zhang L., Miranskyy A., Rjaibi W. Quantum Advantage and the Y2K Bug: A Comparison. *IEEE Software*. 2021. Vol. 38, Issue 2. P. 80–87. DOI: 10.1109/MS.2020.2985321.

136. Zhang Y., Tan X., Qiu T. Quantum violation of LGI under an energy constraint for different scenarios systems. *Scientific Reports*. 2023. Vol. 13, Issue 1. P. 12530. DOI: 10.1038/s41598-023-39612-6.

ДОДАТКИ

Додаток А

ЛІСТИНГ ПРОГРАМНОГО КОДУ

math_gaspa.py

```

import numpy as np
from math import cos, sin, pi
from itertools import product

# =====
# Утиліти: базис, тензорні добутки та красивий друк
# =====

def kron_n(ops):
    """Кронекерів добuток списку матриць (зліва направо)."""
    out = ops[0]
    for op in ops[1:]:
        out = np.kron(out, op)
    return out

def ket0(n: int) -> np.ndarray:
    """Вектор-стовпець  $|0\dots0\rangle$  розміру  $2^n$ ."""
    v = np.zeros((2**n, 1), dtype=complex)
    v[0, 0] = 1.0
    return v

def bitstring(index: int, n: int) -> str:
    """Двійковий рядок для індексу базису у порядку  $|000\rangle, |001\rangle, \dots, |111\rangle$  (q2
q1 q0)."""
    return format(index, f"0{n}b")

def probs_from_state(state: np.ndarray) -> np.ndarray:
    """Ймовірності вимірювання в обчислювальному базисі."""
    amp = state.flatten()
    return np.abs(amp) ** 2

# =====
# Однокубітні вентиля
# =====

I2 = np.eye(2, dtype=complex)
H = (1/np.sqrt(2)) * np.array([[1, 1],
                               [1, -1]], dtype=complex)

def Ry(theta: float) -> np.ndarray:
    """Однокубітний венти́ль  $R_y(\theta)$ ."""
    c = cos(theta/2)
    s = sin(theta/2)
    return np.array([[c, -s],
                     [s, c]], dtype=complex)

def embed_1q(U: np.ndarray, n: int, j: int) -> np.ndarray:
    """
    Вбудовує однокубітний оператор U у n-кубітний регістр, що діє на кубіт j.

    Конвенція (відповідає порядку базису  $|q_{n-1}\dots q_0\rangle$ ):
    - кубіт 0 – найменш значущий (найправіший) біт,
    - порядок тензорів  $[q_{n-1}, \dots, q_0]$ .
    """

```

```

ops = []
for k in range(n-1, -1, -1):
    ops.append(U if k == j else I2)
return kron_n(ops)

# =====
# Матриця контрольованого НЕ для n кубітів: CX_{i -> j}
# =====

def CX(n: int, control: int, target: int) -> np.ndarray:
    """
    Будує повну матрицю розміру 2^n x 2^n для CNOT з заданою керуючою та
    цільовою кубітом.

    Порядок базису: |q_{n-1}...q_0> з індексом = двійкове значення (q_{n-1} -
    старший біт).
    """
    dim = 2**n
    U = np.zeros((dim, dim), dtype=complex)

    for x in range(dim):
        bits = list(map(int, bitstring(x, n))) # біти у порядку [q_{n-
1}, ..., q_0]
        # Перетворює індекс кубіта (0..n-1) на позицію в цьому списку:
        # q_k відповідає bits[n-1-k]
        cpos = n - 1 - control
        tpos = n - 1 - target

        y_bits = bits.copy()
        if bits[cpos] == 1:
            y_bits[tpos] ^= 1

        y = int("".join(map(str, y_bits)), 2)
        U[y, x] = 1.0

    return U

# =====
# Цільова функція Max-Cut для трикутного графу
# E = {(0,1), (0,2), (1,2)}
# =====

E_triangle = [(0, 1), (0, 2), (1, 2)]

def cut_value(bitstr: str, edges) -> int:
    """
    Обчислює C(x) = кількість ребер із кінцевими точками в різних розділах.
    bitstr у порядку q_{n-1}...q_0; вершина v відповідає кубіту v (q_v).
    """
    n = len(bitstr)

    def x_v(v: int) -> int:
        # кубіт v знаходиться на позиції n-1-v у рядку q_{n-1}...q_0
        return int(bitstr[n - 1 - v])

    C = 0
    for (u, v) in edges:
        C += 1 if (x_v(u) ^ x_v(v)) == 1 else 0
    return C

# =====
# Відтворення обчислення з прикладу дисертації
# =====

```

```

n = 3
theta = pi / 4 # example angle

# Початковий стан та шар Адамара
psi0 = ket0(n)
Hn = kron_n([H] * n)
psi1 = Hn @ psi0

# Визначає оператори ребер  $U_{i \rightarrow j}(\theta) = CX * Ry^j(\theta) * CX$ 
def U_edge(n: int, i: int, j: int, theta: float) -> np.ndarray:
    CXij = CX(n, i, j)
    Ryj = embed_1q(Ry(theta), n, j)
    return CXij @ Ryj @ CXij

# Застосовує ребра трикутника у виборі порядку: (0,1) потім (0,2) потім (1,2)
U01 = U_edge(n, 0, 1, theta)
psi2 = U01 @ psi1

U02 = U_edge(n, 0, 2, theta)
psi3 = U02 @ psi2

U12 = U_edge(n, 1, 2, theta)
psi4 = U12 @ psi3

# Ймовірності після повної послідовності ребер
P = probs_from_state(psi4)

# =====
# Виведення результатів (стани, ймовірності та значення Max-Cut)
# =====

np.set_printoptions(precision=6, suppress=True)

print("theta = pi/4")
print("\nСтан |psi1> (після  $H^{\otimes n}$ ):")
print(psi1.real.flatten())

print("\nСтан |psi2> (після  $U_{0 \rightarrow 1}$ ):")
print(psi2.real.flatten())

print("\nСтан |psi3> (після  $U_{0 \rightarrow 2}$ ):")
print(psi3.real.flatten())

print("\nСтан |psi4> (після  $U_{1 \rightarrow 2}$ ):")
print(psi4.real.flatten())

print("\nЙмовірності  $P(x) = |\langle x | \psi_4 \rangle|^2$ :")
for idx, p in enumerate(P):
    bs = bitstring(idx, n)
    print(f"P({bs}) = {p:.6f}    C({bs}) = {cut_value(bs, E_triangle)}")

print("\nПеревірка нормалізації  $\sum_x P(x) =$ ", float(P.sum()))

# Визначає найймовірніші двійкові рядки та їх значення розрізу
top_k = 8
order = np.argsort(-P)[:top_k]
print("\nТоп результатів:")
for idx in order:
    bs = bitstring(int(idx), n)
    print(f"{bs}: P={P[idx]:.6f}, C={cut_value(bs, E_triangle)}")

```

brute_force.py

```
import numpy as np
```

```

from itertools import product
import networkx as nx

# Функція підрахунку ваги розрізу на основі бітового рядка

def brute_force(G, debug=False):
    def compute_cut_value(bitstring, graph):
        cut_value = 0
        n = len(graph)
        for i in range(n):
            for j in range(i + 1, n):
                if graph[i][j] > 0 and bitstring[i] != bitstring[j]:
                    cut_value += graph[i][j]
        return cut_value

    # Функція для вирішення Max-Cut повним перебором

    def max_cut_brute_force(graph):
        num_nodes = len(graph)
        best_cut = 0
        best_solution = None

        # перебір всіх можливих розбиттів
        for bits in product([0, 1], repeat=num_nodes):
            current_cut = compute_cut_value(bits, graph)
            if current_cut > best_cut:
                best_cut = current_cut
                best_solution = bits

        return best_cut, best_solution

    graph = nx.to_numpy_array(G)

    # Виконання алгоритму
    best_cut, best_solution = max_cut_brute_force(graph)

    if debug:
        print("Best cut value:", best_cut)
        print("Best solution:", best_solution)

    return best_cut, best_solution

```

max_cut_qaspa.py

```

from qiskit import QuantumCircuit, transpile

from qiskit.visualization import plot_histogram
import numpy as np
from qiskit_aer import AerSimulator
import networkx as nx
import matplotlib.pyplot as plt

try:
    from ibm import run_on_cloud
except ImportError:
    run_on_cloud = None

def compute_cut_value(bitstring, graph):
    cut_value = 0
    n = len(graph)
    for i in range(n):

```

```

        for j in range(i + 1, n):
            if graph[i][j] > 0:
                if bitstring[i] != bitstring[j]:
                    cut_value += graph[i][j]
    return cut_value

def normalize_angle(value):
    return np.pi / 6

def build_angle_graph(graph):
    angle_graph = np.zeros_like(graph, dtype=float)

    for i in range(len(graph)):
        for j in range(len(graph)):
            angle_graph[i][j] = normalize_angle(
                graph[i][j])
    return angle_graph

def gaspa_maxcut(graph, angle, deep=1):
    num_nodes = len(graph)
    angle_graph = build_angle_graph(graph)
    qc = QuantumCircuit(num_nodes, num_nodes)

    for i in range(num_nodes):
        qc.h(i)

    for _ in range(deep):
        for i in range(num_nodes):
            for j in range(i, num_nodes):
                if graph[i][j] > 0:
                    # angle = angle_graph[i][j]
                    qc.cx(i, j)
                    qc.ry(angle, j)
                    qc.cx(i, j)
                    qc.barrier()

    qc.barrier()
    qc.measure_all()

    return qc

def run_gaspa_maxcut_local(G, debug=False, angle=np.pi/4, deep=1):
    graph = nx.to_numpy_array(G, dtype=int)
    n = len(graph)
    shots = int(np.ceil(0.03047 * (2 ** n) + 1016.2))

    qc = gaspa_maxcut(graph, angle, deep=deep)

    simulator = AerSimulator()
    compiled = transpile(qc, simulator)
    result = simulator.run(compiled, shots=shots).result()
    counts = result.get_counts()

    # Get top 4 solutions
    sorted_counts = sorted(
        counts.items(), key=lambda item: item[1], reverse=True)
    top_4_results = []

    for i, (bitstring, count) in enumerate(sorted_counts[:4]):

```

```

        solution = bitstring.split(' ')[0][::-1]
        cut_value = compute_cut_value(solution, graph)
        top_4_results.append({
            'rank': i + 1,
            'cut_value': cut_value,
            'solution': solution,
            'count': count
        })

    # Best result is the first one
    top_4_results.sort(key=lambda x: x['cut_value'], reverse=True)
    best_cut = top_4_results[0]['cut_value']
    best_solution = top_4_results[0]['solution']

    if debug:
        print("Number of shots:", shots)
        # print("Final circuit:\n", qc)
        # print("Counts:\n", dict(sorted(counts.items(), key=lambda item:
item[1], reverse=True)))
        print("\n=== TOP 4 SOLUTIONS ===")
        for result in top_4_results:
            print(
                f"Rank {result['rank']}: Cut = {result['cut_value']},
Solution = {result['solution']}, Count = {result['count']}")
            print(f"\nBest solution: {best_solution[::-1]}")
            print(f"Best cut value: {best_cut}")
            colors = ['r' if best_solution[node] == '0' else 'b' for node in G]
            nx.draw(G, with_labels=False, node_color=colors)
            plt.savefig("qaspa_maxcut1.png")

            res = {k.split(' ')[0]: v for k, v in counts.items()}
            plot_histogram(res, bar_labels=False, legend=False).savefig(
                "qaspa_maxcut_counts1.png")

    return best_cut, best_solution, dict(sorted(counts.items(), key=lambda
item: item[1], reverse=True))

def run_qaspa_maxcut_cloud(G, debug=False, shots=1024, deep=1):
    if run_on_cloud is None:
        raise ImportError(
            "IBM Runtime not available. Please install qiskit_ibm_runtime.")

    graph = nx.to_numpy_array(G, dtype=int)
    qc = qaspa_maxcut(graph, angle=np.pi/6, deep=deep)

    counts = run_on_cloud(qc, debug, shots)

    # Get top 4 solutions
    sorted_counts = sorted(
        counts.items(), key=lambda item: item[1], reverse=True)
    top_4_results = []

    for i, (bitstring, count) in enumerate(sorted_counts[:4]):
        solution = bitstring[::-1]
        cut_value = compute_cut_value(solution, graph)
        top_4_results.append({
            'rank': i + 1,
            'cut_value': cut_value,
            'solution': solution,
            'count': count
        })

    # Best result is the first one

```

```

cloud_best_cut = top_4_results[0]['cut_value']
cloud_best_solution = top_4_results[0]['solution']

if debug:
    print("Final circuit:\n", qc)
    print("Counts:\n", dict(sorted(counts.items(),
                                   key=lambda item: item[1], reverse=True)))
    print("\n=== TOP 4 SOLUTIONS ===")
    for result in top_4_results:
        print(
            f"Rank {result['rank']}: Cut = {result['cut_value']},
Solution = {result['solution']}, Count = {result['count']}"
        )
    print(f"\nBest solution: {cloud_best_solution[:-1]}")
    print(f"Best cut value: {cloud_best_cut}")
    colors = ['r' if cloud_best_solution[node]
              == '0' else 'b' for node in G]
    nx.draw(G, with_labels=True, node_color=colors)
    plt.savefig("qaspa_maxcut_cloud.png")

return cloud_best_cut, cloud_best_solution, top_4_results

```

ibm.py

```

from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2 as Sampler
from qiskit.transpiler import generate_preset_pass_manager

from qiskit.circuit import QuantumCircuit
from qiskit.visualization import plot_histogram
import numpy as np

def run_on_cloud(circuit, debug=False, shots=1024):

    service = QiskitRuntimeService(token='TOKEN', channel='ibm_cloud',
instance='test')

    backend = service.backend('ibm_torino')

    pm = generate_preset_pass_manager(optimization_level=1, backend=backend)
    isa_circuit = pm.run(circuit)
    param_values = np.random.rand(circuit.num_parameters)

    # execute the quantum circuit
    sampler = Sampler(backend)
    job = sampler.run([(isa_circuit, param_values)], shots=shots)
    result = job.result()
    pub_result = result[0]
    counts = pub_result.data.meas.get_counts()
    if debug:
        print("Transpiled circuit: -----")
        print(isa_circuit)
        print("Result: -----")
        print(result)
        print("-----")
        print("Counts: -----")
        print(counts)
        print("-----")
        plot_histogram(counts).savefig("ibm_result.png")

```

```

    print("File ibm_result.png saved")

    return counts

```

qaoa_qrisp.py

```

from qrisp.algorithms.qaoa import QuantumArray, QuantumVariable, QAOAProblem,
maxcut_obj, create_maxcut_cl_cost_function, create_maxcut_cost_operator,
RX_mixer
import networkx as nx
from operator import itemgetter
import matplotlib.pyplot as plt

def qaoa_qrisp(G, debug=False):

    qarg = QuantumArray(qtype=QuantumVariable(1), shape=len(G))

    depth = 1

    maxcut_instance = QAOAProblem(create_maxcut_cost_operator(
        G), RX_mixer, create_maxcut_cl_cost_function(G))

    res = maxcut_instance.run(qarg, depth, max_iter=10)

    res_str = list(res.keys())[0]

    value = ''.join([str(int(x)) for x in res_str])

    best_cut, best_solution = (maxcut_obj(value, G), res_str)

    if debug:
        print("Best cut value:", best_cut)
        print("Best solution:", best_solution)
        print("Result:", res)
        print("Result value:", value)
        # colors = ['r' if best_solution[node] == '0' else 'b' for node in G]
        # nx.draw(G, with_labels=True, node_color=colors)
        # plt.savefig("qaoa_qrisp.png")

        plt.figure()
        plt.bar(res.keys(), res.values())
        plt.xlabel('Solutions')
        plt.ylabel('Probabilities')
        plt.title('QAOA Result Histogram')
        plt.xticks(rotation=90)
        plt.tight_layout()
        plt.savefig("qaoa_qrisp_histogram.png")

    return abs(best_cut), best_solution

```

tests.py

```

"""
Test module for MaxCut algorithms.
Tests are organized using test_samples module for better maintainability.
"""

from max_cut_gaspa import run_gaspa_maxcut_local, run_gaspa_maxcut_cloud
from utils import assertEqual, save_graph

```

```

from qaoa_qrisp import qaoa_qrisp
from brute_force import brute_force
from qaoa_qiskit import QAOA_maxcut_qiskit
import networkx as nx
import numpy as np
import time
import os
import matplotlib.pyplot as plt

from test_samples import get_test_sample, get_all_test_samples,
print_test_samples_info

def cycleTest(index):
    G = nx.erdos_renyi_graph(index, 0.3, seed=42)
    save_graph(G, f"erdos_renyi_graph_{index}")

    # Get brute force optimal solution
    (brute_force_best_cut, _) = brute_force(G, debug=True)

    # Run 100 times and collect approximation ratios
    approximation_ratios = []
    correct_count = 0

    for trial in range(1):
        best_cut, _, _ = run_gaspa_maxcut_local(
            G, debug=True, angle=np.pi/4, deep=1)

        # Calculate approximation ratio
        ratio = best_cut / brute_force_best_cut if brute_force_best_cut > 0
    else 0
        approximation_ratios.append(ratio)

        # Count successful runs (ratio >= 0.9)
        if ratio >= 0.9:
            correct_count += 1

    # Calculate statistics
    mean_ratio = np.mean(approximation_ratios)
    median_ratio = np.median(approximation_ratios)
    std_ratio = np.std(approximation_ratios)
    min_ratio = np.min(approximation_ratios)
    max_ratio = np.max(approximation_ratios)
    success_rate = (correct_count / 10) * 100

    # Print results
    print(f"\nGraph size {index}:")
    print(f"  Brute force optimal cut: {brute_force_best_cut}")
    print(f"  Mean approximation ratio: {mean_ratio:.4f}")
    print(f"  Median approximation ratio: {median_ratio:.4f}")
    print(f"  Std deviation: {std_ratio:.4f}")
    print(f"  Range: [{min_ratio:.4f}, {max_ratio:.4f}]")
    print(f"  Success rate: {success_rate:.1f}% ({correct_count}/100)")

def cycleTitmeTest(index):
    G = nx.complete_graph(index)

    start_time = time.time()
    (best_cut, best_solution) = run_gaspa_maxcut_local(G, debug=False)
    gaspa_time = time.time() - start_time

    start_time = time.time()
    (qaoa_best_cut, best_solution) = qaoa_qrisp(G, debug=False)

```

```

qaoa_time = time.time() - start_time

start_time = time.time()
(brute_force_best_cut, best_solution) = brute_force(G)
brute_time = time.time() - start_time

return best_cut, qaspa_time, qaoa_time, brute_time

def plot_times():
    indices = range(10, 20) # Adjust the range as needed
    qaspa_times = []
    qaoa_times = []
    brute_force_times = []

    for i in indices:
        _, qaspa_time, qaoa_time, brute_time = cycleTitmeTest(i)
        print(
            f"Graph size {i}: QASPA time = {qaspa_time:.4f}s, QAOA time =
{qaoa_time:.4f}s, Brute Force time = {brute_time:.4f}s")
        qaspa_times.append(qaspa_time)
        qaoa_times.append(qaoa_time)
        brute_force_times.append(brute_time)

    plt.figure(figsize=(10, 6))
    plt.plot(indices, qaspa_times, label="QASPA MaxCut", marker='o')
    plt.plot(indices, qaoa_times, label="QAOA QRISP", marker='o')
    plt.plot(indices, brute_force_times, label="Brute Force", marker='o')
    plt.xlabel("Graph Size (Number of Nodes)")
    plt.ylabel("Execution Time (seconds)")
    plt.title("Execution Time vs Graph Size")
    plt.legend()
    plt.grid()
    plt.savefig("execution_time.png")

#
=====
# MANUAL TESTS - Using test samples from test_samples module
#
=====

def run_single_manual_test(test_name, debug=False):

    sample = get_test_sample(test_name)
    if sample is None:
        print(f"Test '{test_name}' not found!")
        return

    G = sample.build_graph()
    if debug:
        save_graph(G, f"{test_name}_graph")

    print(f"\n{'='*70}")
    print(f"Running: {test_name}")
    print(f>Description: {sample.description}")
    print(f"Nodes: {G.number_of_nodes()}, Edges: {G.number_of_edges()}")
    print(f"{'='*70}")

    # Run all algorithms
    (qaspa_cut, qaspa_solution, _) = run_qaspa_maxcut_local(
        G, debug=debug, angle=np.pi/4, deep=1)
    (brute_force_cut, brute_force_solution) = brute_force(G)
    (qaoa_cut, qaoa_solution) = qaoa_qrisp(G, debug=debug)

```

```

# Print results
if debug:
    print(f"\nResults:")
    print(
        f"   Brute Force: Cut={brute_force_cut},
Solution={brute_force_solution}")
    print(f"   QASPA:      Cut={qaspa_cut}, Solution={qaspa_solution}")
    print(f"   QAOA QRisp:  Cut={qaoa_cut}, Solution={qaoa_solution}")

# Check correctness using assertEquals
try:
    assertEquals(test_name + " QASPA", qaspa_cut, brute_force_cut, )
    assertEquals(test_name + " QAOA", qaoa_cut, brute_force_cut, )
    print(f"\n")
except AssertionError as e:
    ratio = qaspa_cut / brute_force_cut if brute_force_cut > 0 else 0
    print(f"\nX {e} (Ratio: {ratio:.4f})")

return qaspa_cut, brute_force_cut, qaoa_cut

def test1():
    """Test: Square with center node"""
    run_single_manual_test('test1')

def test2():
    """Test: Triangle"""
    run_single_manual_test('test2')

def test3():
    """Test: Square"""
    run_single_manual_test('test3')

def test4():
    """Test: Path graph (5 nodes)"""
    run_single_manual_test('test4')

def test5():
    """Test: Square with diagonal"""
    run_single_manual_test('test5')

def test6():
    """Test: Pentagon"""
    run_single_manual_test('test6')

def test7():
    """Test: Path graph (4 nodes)"""
    run_single_manual_test('test7')

def test8():
    """Test: Path graph (6 nodes)"""
    run_single_manual_test('test8')

def test9():

```

```

        """Test: Triangle with tail"""
        run_single_manual_test('test9')

def test10():
    """Test: Pentagon with diagonal"""
    run_single_manual_test('test10')

def test11():
    """Test: Hexagon"""
    run_single_manual_test('test11')

def test12():
    """Test: Triangle with tail"""
    run_single_manual_test('test12')

def test13():
    """Test: Hexagon"""
    run_single_manual_test('test13')

def test14():
    """Test: Path with additional edge"""
    run_single_manual_test('test14')

def test15():
    """Test: Path with chord"""
    run_single_manual_test('test15')

def test16():
    """Test: Path with chord"""
    run_single_manual_test('test16')

def test17():
    """Test: Complex graph"""
    run_single_manual_test('test17')

#
=====
# MAIN TEST EXECUTION
#
=====

def tests():
    test1()
    test2()
    test3()
    test4()
    test5()
    test6()
    test7()
    test8()
    test9()
    test10()
    test11()
    test12()
    test13()

```

```

    test14()
    test15()
    test16()
    test17()
tests()
utils.py

import matplotlib.pyplot as plt
import networkx as nx
import os

def assertEqual(name, a, b):
    if a == b:
        print(f"\033[92m{name}: Test passed!\033[0m")
        return True
    else:
        print(
            f"\033[91m{name}: Test failed!\n\tActual result is
{a}\n\tExpected result is {b}\033[0m")
        return False

def save_graph(G, filename, folder="graphs_output"):
    """
    Saves a NetworkX graph as an image.

    Args:
        G: NetworkX graph object
        filename: Name of the output file (without extension)
        folder: Folder to save the graph (default: 'graphs_output')
    """
    # Create folder if it doesn't exist
    if not os.path.exists(folder):
        os.makedirs(folder)

    # Create figure
    plt.figure(figsize=(10, 8))

    # Draw the graph
    pos = nx.spring_layout(G, seed=42)
    nx.draw_networkx_nodes(G, pos, node_color='lightblue', node_size=500)
    nx.draw_networkx_edges(G, pos, width=2)
    nx.draw_networkx_labels(G, pos, font_size=10, font_weight='bold')

    plt.title(f"Graph: {filename}")
    plt.axis('off')

    # Save the figure
    filepath = os.path.join(folder, f"{filename}.png")
    plt.savefig(filepath, dpi=100, bbox_inches='tight')
    plt.close()

    print(f"Graph saved to {filepath}")
test_samples.py
"""
Test samples module containing predefined graph configurations for MaxCut
testing.
This module provides a structured way to define and manage test cases.
"""

import networkx as nx

class GraphSample:

```

```

"""Represents a single test graph sample."""

def __init__(self, name, edges, description=""):
    """
    Initialize a graph sample.

    Args:
        name: Test name (e.g., "test1")
        edges: List of tuples representing edges
        description: Human-readable description of the graph
    """
    self.name = name
    self.edges = edges
    self.description = description

def build_graph(self):
    """Build and return NetworkX graph from edges."""
    G = nx.Graph()
    G.add_edges_from(self.edges)
    return G

def get_node_count(self):
    """Get number of nodes in the graph."""
    G = self.build_graph()
    return G.number_of_nodes()

def get_edge_count(self):
    """Get number of edges in the graph."""
    return len(self.edges)

def __repr__(self):
    return f"GraphSample({self.name}: {self.get_node_count()} nodes, {self.get_edge_count()} edges)"

# Define all test samples
TEST_SAMPLES = [
    GraphSample(
        "test1",
        [(0, 1), (1, 2), (2, 3), (3, 0), (0, 4), (1, 4), (2, 4), (3, 4)],
        "Square with center node (8 edges)"
    ),
    GraphSample(
        "test2",
        [(0, 1), (1, 2), (2, 0)],
        "Triangle (3 edges)"
    ),
    GraphSample(
        "test3",
        [(0, 1), (1, 2), (2, 3), (3, 0)],
        "Square (4 edges)"
    ),
    GraphSample(
        "test4",
        [(0, 1), (1, 2), (2, 3), (3, 4)],
        "Path graph (5 nodes, 4 edges)"
    ),
    GraphSample(
        "test5",
        [(0, 1), (1, 2), (2, 3), (3, 0), (0, 2)],
        "Square with diagonal (5 edges)"
    ),
    GraphSample(
        "test6",

```

```

        [(0, 1), (1, 2), (2, 3), (3, 4), (4, 0)],
        "Pentagon (5 nodes, 5 edges)"
    ),
    GraphSample(
        "test7",
        [(0, 1), (1, 2), (2, 3)],
        "Path graph (4 nodes, 3 edges)"
    ),
    GraphSample(
        "test8",
        [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5)],
        "Path graph (6 nodes, 5 edges)"
    ),
    GraphSample(
        "test9",
        [(0, 1), (1, 2), (2, 0), (2, 3)],
        "Triangle with tail (4 nodes, 4 edges)"
    ),
    GraphSample(
        "test10",
        [(0, 1), (1, 2), (2, 3), (3, 4), (4, 0), (1, 3)],
        "Pentagon with diagonal (5 nodes, 6 edges)"
    ),
    GraphSample(
        "test11",
        [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 0)],
        "Hexagon (6 nodes, 6 edges)"
    ),
    GraphSample(
        "test12",
        [(0, 1), (1, 2), (2, 0), (2, 3)],
        "Triangle with tail (4 nodes, 4 edges)"
    ),
    GraphSample(
        "test13",
        [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 0)],
        "Hexagon (6 nodes, 6 edges)"
    ),
    GraphSample(
        "test14",
        [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (0, 3)],
        "Path with additional edge (6 nodes, 6 edges)"
    ),
    GraphSample(
        "test15",
        [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (1, 4)],
        "Path with chord (6 nodes, 6 edges)"
    ),
    GraphSample(
        "test16",
        [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (2, 5)],
        "Path with chord (6 nodes, 6 edges)"
    ),
    GraphSample(
        "test17",
        [(0, 1), (0, 2), (0, 6), (1, 2), (1, 3), (1, 6),
         (2, 6), (2, 5), (3, 6), (3, 5), (3, 4), (4, 6), (4, 5), (5, 6)],
        "Complex graph (7 nodes, 14 edges)"
    ),
]

```

```

def get_test_sample(test_name):
    """

```

Get a test sample by name.

Args:

test_name: Name of the test (e.g., 'test1')

Returns:

GraphSample object or None if not found

```
"""
for sample in TEST_SAMPLES:
    if sample.name == test_name:
        return sample
return None
```

```
def get_all_test_samples():
    """Get all test samples."""
    return TEST_SAMPLES
```

```
def print_test_samples_info():
    """Print information about all test samples."""
    print("\n" + "="*80)
    print("TEST SAMPLES INFORMATION")
    print("="*80)
    for i, sample in enumerate(TEST_SAMPLES, 1):
        print(f"{i:2d}. {sample.name:8s} - {sample.description:40s} |
{sample.get_node_count()} nodes, {sample.get_edge_count()} edges")
    print("="*80 + "\n")
```

run_manual_tests.py

```
"""
Manual test runner for MaxCut algorithms.
Compares performance of QASPA, QAOA (QRisp), and Brute Force algorithms.
Generates comparison tables and visualizations.
"""
```

```
import networkx as nx
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
import os
from pathlib import Path
```

```
from max_cut_gaspa import run_gaspa_maxcut_local
from qaoa_qrisp import qaoa_qrisp
from brute_force import brute_force
from utils import save_graph
from test_samples import get_all_test_samples, print_test_samples_info
```

```
class TestResult:
    """Store results of a single test run."""

    def __init__(self, test_name, num_nodes, num_edges):
        self.test_name = test_name
        self.num_nodes = num_nodes
        self.num_edges = num_edges

        # Results
        self.brute_force_cut = None
        self.brute_force_time = None
        self.brute_force_solution = None
```

```

        self.gaspa_cut = None
        self.gaspa_time = None
        self.gaspa_solution = None

        self.qaoa_cut = None
        self.qaoa_time = None
        self.qaoa_solution = None

    def get_approximation_ratios(self):
        """Calculate approximation ratios relative to brute force."""
        if self.brute_force_cut is None or self.brute_force_cut == 0:
            return None, None

        gaspa_ratio = self.gaspa_cut / self.brute_force_cut if self.gaspa_cut
    else 0
        qaoa_ratio = self.qaoa_cut / self.brute_force_cut if self.qaoa_cut
    else 0

        return gaspa_ratio, qaoa_ratio

    def to_dict(self):
        """Convert to dictionary for DataFrame."""
        gaspa_ratio, qaoa_ratio = self.get_approximation_ratios()

        return {
            'Test': self.test_name,
            'Nodes': self.num_nodes,
            'Edges': self.num_edges,
            'BF Cut': self.brute_force_cut,
            'BF Time (s)': self.brute_force_time,
            'QASPA Cut': self.gaspa_cut,
            'QASPA Time (s)': self.gaspa_time,
            'QASPA Ratio': gaspa_ratio,
            'QAOA Cut': self.qaoa_cut,
            'QAOA Time (s)': self.qaoa_time,
            'QAOA Ratio': qaoa_ratio,
        }

class TestRunner:
    """Runner for MaxCut algorithm tests."""

    def __init__(self, verbose=True, save_graphs=False):
        self.verbose = verbose
        self.save_graphs = save_graphs
        self.results = []

    def run_test(self, graph_sample):
        """
        Run test for a single graph sample.

        Args:
            graph_sample: GraphSample object

        Returns:
            TestResult object
        """
        G = graph_sample.build_graph()
        result = TestResult(
            graph_sample.name,
            G.number_of_nodes(),
            G.number_of_edges()
        )

```

```

    if self.verbose:
        print(f"\n{'='*70}")
        print(f"Testing: {graph_sample.name}")
        print(f"Description: {graph_sample.description}")
        print(f"Nodes: {G.number_of_nodes()}, Edges:
{G.number_of_edges()}")
        print(f"{'='*70}")

    # Save graph if requested
    if self.save_graphs:
        save_graph(G, f"{graph_sample.name}_graph")

    # Run Brute Force
    try:
        start = time.time()
        result.brute_force_cut, result.brute_force_solution =
brute_force(G, debug=False)
        result.brute_force_time = time.time() - start
        if self.verbose:
            print(f"  Brute Force:  Cut={result.brute_force_cut:4d},
Time={result.brute_force_time:.4f}s")
        except Exception as e:
            print(f"  Brute Force ERROR: {str(e)}")
            return None

    # Run QASPA
    try:
        start = time.time()
        result.qaspa_cut, result.qaspa_solution, _ =
run_qaspa_maxcut_local(
            G, debug=False, angle=np.pi/4, deep=1
        )
        result.qaspa_time = time.time() - start
        if self.verbose:
            print(f"  QASPA:          Cut={result.qaspa_cut:4d},
Time={result.qaspa_time:.4f}s")
        except Exception as e:
            print(f"  QASPA ERROR: {str(e)}")
            result.qaspa_cut = 0
            result.qaspa_time = 0

    # Run QAOA QRisp
    try:
        start = time.time()
        result.qaoa_cut, result.qaoa_solution = qaoa_qrisp(G,
debug=False)
        result.qaoa_time = time.time() - start
        if self.verbose:
            print(f"  QAOA QRisp:    Cut={result.qaoa_cut:4d},
Time={result.qaoa_time:.4f}s")
        except Exception as e:
            print(f"  QAOA QRisp ERROR: {str(e)}")
            result.qaoa_cut = 0
            result.qaoa_time = 0

    # Show approximation ratios
    qaspa_ratio, qaoa_ratio = result.get_approximation_ratios()
    if qaspa_ratio is not None:
        if self.verbose:
            print(f"\n  Approximation Ratios:")
            print(f"    QASPA: {qaspa_ratio:.4f}")
            print(f"    QAOA: {qaoa_ratio:.4f}")

    self.results.append(result)

```

```

        return result

def run_all_tests(self, samples=None):
    """
    Run tests for all or specified samples.

    Args:
        samples: List of GraphSample objects. If None, runs all.
    """
    if samples is None:
        samples = get_all_test_samples()

    print_test_samples_info()

    for sample in samples:
        self.run_test(sample)

def get_results_dataframe(self):
    """Get results as pandas DataFrame."""
    data = [r.to_dict() for r in self.results]
    return pd.DataFrame(data)

def print_results_table(self):
    """Print results as formatted table."""
    df = self.get_results_dataframe()

    print("\n" + "="*120)
    print("DETAILED RESULTS TABLE")
    print("="*120)
    print(df.to_string(index=False))
    print("="*120 + "\n")

def print_summary_table(self):
    """Print summary table comparing algorithms."""
    df = self.get_results_dataframe()

    summary = pd.DataFrame({
        'Test': df['Test'],
        'Nodes': df['Nodes'],
        'Edges': df['Edges'],
        'BF': df['BF Cut'],
        'QASPA': df['QASPA Cut'],
        'QASPA Ratio': df['QASPA Ratio'].round(4),
        'QAOA': df['QAOA Cut'],
        'QAOA Ratio': df['QAOA Ratio'].round(4),
    })

    print("\n" + "="*100)
    print("SUMMARY: ALGORITHM COMPARISON")
    print("="*100)
    print(summary.to_string(index=False))
    print("="*100 + "\n")

def print_timing_table(self):
    """Print timing comparison table."""
    df = self.get_results_dataframe()

    timing = pd.DataFrame({
        'Test': df['Test'],
        'Nodes': df['Nodes'],
        'BF Time (s)': df['BF Time (s)'].round(6),
        'QASPA Time (s)': df['QASPA Time (s)'].round(6),
        'QAOA Time (s)': df['QAOA Time (s)'].round(6),
    })

```

```

print("\n" + "="*80)
print("TIMING COMPARISON")
print("="*80)
print(timing.to_string(index=False))
print("="*80 + "\n")

def plot_cut_comparison(self, save_path='plots/cut_comparison.png'):
    """Plot cut quality comparison."""
    df = self.get_results_dataframe()

    fig, ax = plt.subplots(figsize=(14, 6))

    x = np.arange(len(df))
    width = 0.25

    ax.bar(x - width, df['BF Cut'], width, label='Brute Force',
alpha=0.8, color='#1f77b4')
    ax.bar(x, df['QASPA Cut'], width, label='QASPA', alpha=0.8,
color='#ff7f0e')
    ax.bar(x + width, df['QAOA Cut'], width, label='QAOA QRisp',
alpha=0.8, color='#2ca02c')

    ax.set_xlabel('Test Case', fontsize=12, fontweight='bold')
    ax.set_ylabel('Cut Size', fontsize=12, fontweight='bold')
    ax.set_title('MaxCut Solution Comparison: Cut Quality', fontsize=14,
fontweight='bold')
    ax.set_xticks(x)
    ax.set_xticklabels(df['Test'], rotation=45, ha='right')
    ax.legend(fontsize=11)
    ax.grid(axis='y', alpha=0.3)

    plt.tight_layout()
    self._ensure_dir(save_path)
    plt.savefig(save_path, dpi=300, bbox_inches='tight')
    if self.verbose:
        print(f"Saved plot: {save_path}")
    plt.close()

def plot_approximation_ratios(self,
save_path='plots/approximation_ratios.png'):
    """Plot approximation ratios."""
    df = self.get_results_dataframe()

    fig, ax = plt.subplots(figsize=(12, 6))

    x = np.arange(len(df))
    width = 0.35

    ax.bar(x - width/2, df['QASPA Ratio'], width, label='QASPA',
alpha=0.8, color='#ff7f0e')
    ax.bar(x + width/2, df['QAOA Ratio'], width, label='QAOA QRisp',
alpha=0.8, color='#2ca02c')
    ax.axhline(y=1.0, color='r', linestyle='--', linewidth=2,
label='Optimal (1.0)')

    ax.set_xlabel('Test Case', fontsize=12, fontweight='bold')
    ax.set_ylabel('Approximation Ratio', fontsize=12, fontweight='bold')
    ax.set_title('Algorithm Approximation Ratios vs Brute Force',
fontsize=14, fontweight='bold')
    ax.set_xticks(x)
    ax.set_xticklabels(df['Test'], rotation=45, ha='right')
    ax.legend(fontsize=11)
    ax.grid(axis='y', alpha=0.3)

```

```

ax.set_ylim([0, 1.1])

plt.tight_layout()
self._ensure_dir(save_path)
plt.savefig(save_path, dpi=300, bbox_inches='tight')
if self.verbose:
    print(f"Saved plot: {save_path}")
plt.close()

def plot_timing_comparison(self,
save_path='plots/timing_comparison.png'):
    """Plot execution time comparison."""
    df = self.get_results_dataframe()

    fig, ax = plt.subplots(figsize=(14, 6))

    x = np.arange(len(df))
    width = 0.25

    ax.bar(x - width, df['BF Time (s)'], width, label='Brute Force',
alpha=0.8, color='#1f77b4')
    ax.bar(x, df['QASPA Time (s)'], width, label='QASPA', alpha=0.8,
color='#ff7f0e')
    ax.bar(x + width, df['QAOA Time (s)'], width, label='QAOA QRisp',
alpha=0.8, color='#2ca02c')

    ax.set_xlabel('Test Case', fontsize=12, fontweight='bold')
    ax.set_ylabel('Time (seconds)', fontsize=12, fontweight='bold')
    ax.set_title('Algorithm Execution Time Comparison', fontsize=14,
fontweight='bold')
    ax.set_xticks(x)
    ax.set_xticklabels(df['Test'], rotation=45, ha='right')
    ax.legend(fontsize=11)
    ax.grid(axis='y', alpha=0.3)

    plt.tight_layout()
    self._ensure_dir(save_path)
    plt.savefig(save_path, dpi=300, bbox_inches='tight')
    if self.verbose:
        print(f"Saved plot: {save_path}")
    plt.close()

def plot_scalability(self, save_path='plots/scalability.png'):
    """Plot execution time vs graph size."""
    df = self.get_results_dataframe()

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

    # By number of nodes
    df_sorted = df.sort_values('Nodes')
    ax1.plot(df_sorted['Nodes'], df_sorted['BF Time (s)'], 'o-',
linewidth=2, markersize=8, label='Brute Force', color='#1f77b4')
    ax1.plot(df_sorted['Nodes'], df_sorted['QASPA Time (s)'], 's-',
linewidth=2, markersize=8, label='QASPA', color='#ff7f0e')
    ax1.plot(df_sorted['Nodes'], df_sorted['QAOA Time (s)'], '^-',
linewidth=2, markersize=8, label='QAOA QRisp', color='#2ca02c')
    ax1.set_xlabel('Number of Nodes', fontsize=11, fontweight='bold')
    ax1.set_ylabel('Time (seconds)', fontsize=11, fontweight='bold')
    ax1.set_title('Execution Time vs Number of Nodes', fontsize=12,
fontweight='bold')
    ax1.legend(fontsize=10)
    ax1.grid(True, alpha=0.3)

    # By number of edges

```

```

df_sorted = df.sort_values('Edges')
ax2.plot(df_sorted['Edges'], df_sorted['BF Time (s)'], 'o-',
linewidth=2, markersize=8, label='Brute Force', color='#1f77b4')
ax2.plot(df_sorted['Edges'], df_sorted['QASPA Time (s)'], 's-',
linewidth=2, markersize=8, label='QASPA', color='#ff7f0e')
ax2.plot(df_sorted['Edges'], df_sorted['QAOA Time (s)'], '^-',
linewidth=2, markersize=8, label='QAOA QRisp', color='#2ca02c')
ax2.set_xlabel('Number of Edges', fontsize=11, fontweight='bold')
ax2.set_ylabel('Time (seconds)', fontsize=11, fontweight='bold')
ax2.set_title('Execution Time vs Number of Edges', fontsize=12,
fontweight='bold')
ax2.legend(fontsize=10)
ax2.grid(True, alpha=0.3)

plt.tight_layout()
self._ensure_dir(save_path)
plt.savefig(save_path, dpi=300, bbox_inches='tight')
if self.verbose:
    print(f"Saved plot: {save_path}")
plt.close()

def generate_all_plots(self):
    """Generate all plots."""
    print("\n" + "="*70)
    print("GENERATING PLOTS...")
    print("="*70)

    self.plot_cut_comparison()
    self.plot_approximation_ratios()
    self.plot_timing_comparison()
    self.plot_scalability()

    print("="*70 + "\n")

def save_results_to_csv(self, filepath='results/test_results.csv'):
    """Save results to CSV file."""
    df = self.get_results_dataframe()
    self._ensure_dir(filepath)
    df.to_csv(filepath, index=False)
    if self.verbose:
        print(f"Results saved to: {filepath}")

def print_statistics(self):
    """Print statistical summary."""
    df = self.get_results_dataframe()

    print("\n" + "="*70)
    print("STATISTICAL SUMMARY")
    print("="*70)

    gaspa_ratios = df['QASPA Ratio'].dropna()
    qaoa_ratios = df['QAOA Ratio'].dropna()

    print("\nQASPA Approximation Ratios:")
    print(f"  Mean:    {gaspa_ratios.mean():.4f}")
    print(f"  Median:  {gaspa_ratios.median():.4f}")
    print(f"  Std:     {gaspa_ratios.std():.4f}")
    print(f"  Min:     {gaspa_ratios.min():.4f}")
    print(f"  Max:     {gaspa_ratios.max():.4f}")

    print("\nQAOA QRisp Approximation Ratios:")
    print(f"  Mean:    {qaoa_ratios.mean():.4f}")
    print(f"  Median:  {qaoa_ratios.median():.4f}")
    print(f"  Std:     {qaoa_ratios.std():.4f}")

```

```

print(f"   Min:      {qaoa_ratios.min():.4f}")
print(f"   Max:      {qaoa_ratios.max():.4f}")

bf_times = df['BF Time (s)'].dropna()
qaspa_times = df['QASPA Time (s)'].dropna()
qaoa_times = df['QAOA Time (s)'].dropna()

print("\nExecution Times:")
print(f"   Brute Force - Mean: {bf_times.mean():.4f}s, Max:
{bf_times.max():.4f}s")
print(f"   QASPA          - Mean: {qaspa_times.mean():.4f}s, Max:
{qaspa_times.max():.4f}s")
print(f"   QAOA QRisp    - Mean: {qaoa_times.mean():.4f}s, Max:
{qaoa_times.max():.4f}s")

print("="*70 + "\n")

@staticmethod
def _ensure_dir(filepath):
    """Ensure directory exists for filepath."""
    directory = os.path.dirname(filepath)
    if directory and not os.path.exists(directory):
        os.makedirs(directory, exist_ok=True)

def main():
    """Main test runner."""
    print("\n" + "="*70)
    print("MAXCUT ALGORITHM COMPARISON TEST SUITE")
    print("="*70 + "\n")

    # Create runner
    runner = TestRunner(verbose=True, save_graphs=False)

    # Run all tests
    runner.run_all_tests()

    # Print results
    runner.print_summary_table()
    runner.print_timing_table()
    runner.print_results_table()

    # Print statistics
    runner.print_statistics()

    # Generate plots
    runner.generate_all_plots()

    # Save results
    runner.save_results_to_csv()

    print("="*70)
    print("TEST SUITE COMPLETED!")
    print("="*70 + "\n")

if __name__ == "__main__":
    main()

```

run_generated_tests.py

```

"""
Testing framework for generated graphs (Complete, Cycle, Erdős-Rényi).
Compares algorithm performance across different graph types and sizes.
Tracks success rate (approximation ratio >= 0.9) and quality metrics.

```

```

"""

import networkx as nx
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
import os
from pathlib import Path

from max_cut_gaspa import run_gaspa_maxcut_local
from qaoa_grisp import qaoa_grisp
from brute_force import brute_force
from utils import save_graph

# Test configuration
GRAPH_SIZES = list(range(3, 15))
SUCCESS_THRESHOLD = 0.9
BASE_SEED = 42

class GeneratedGraphResult:
    """Store results for a generated graph test."""

    def __init__(self, graph_type, graph_size, trial_num):
        self.graph_type = graph_type
        self.graph_size = graph_size
        self.trial_num = trial_num
        self.num_edges = None

        # Brute Force results
        self.bf_cut = None
        self.bf_time = None

        # QASPA results
        self.gaspa_cut = None
        self.gaspa_time = None
        self.gaspa_ratio = None
        self.gaspa_success = False

        # QAOA results
        self.qaoa_cut = None
        self.qaoa_time = None
        self.qaoa_ratio = None
        self.qaoa_success = False

    def to_dict(self):
        """Convert to dictionary for DataFrame."""
        return {
            'Graph Type': self.graph_type,
            'Size': self.graph_size,
            'Edges': self.num_edges,
            'Trial': self.trial_num,
            'BF Cut': self.bf_cut,
            'BF Time (s)': self.bf_time,
            'QASPA Cut': self.gaspa_cut,
            'QASPA Time (s)': self.gaspa_time,
            'QASPA Ratio': self.gaspa_ratio,
            'QASPA Success': self.gaspa_success,
            'QAOA Cut': self.qaoa_cut,
            'QAOA Time (s)': self.qaoa_time,
            'QAOA Ratio': self.qaoa_ratio,
            'QAOA Success': self.qaoa_success,

```

```

    }

def generate_graphs(graph_size, trial_num):
    """
    Generate different types of graphs for testing.

    Args:
        graph_size: Number of nodes
        trial_num: Trial number for seeding

    Returns:
        Dictionary of graph_type -> NetworkX graph
    """
    graphs = {}
    seed = BASE_SEED

    # 1. Complete graph
    G_complete = nx.complete_graph(graph_size)
    graphs['Complete'] = G_complete

    # 2. Cycle graph
    G_cycle = nx.cycle_graph(graph_size)
    graphs['Cycle'] = G_cycle

    # 3. Erdős-Rényi with p=0.3
    G_er_03 = nx.erdos_renyi_graph(graph_size, 0.3, seed=seed)
    if G_er_03.number_of_edges() == 0:
        G_er_03.add_edge(0, 1)
    graphs['Erdos-Renyi (p=0.3)'] = G_er_03

    # 4. Erdős-Rényi with p=0.5
    G_er_05 = nx.erdos_renyi_graph(graph_size, 0.5, seed=seed)
    if G_er_05.number_of_edges() == 0:
        G_er_05.add_edge(0, 1)
    graphs['Erdos-Renyi (p=0.5)'] = G_er_05

    # 5. Erdős-Rényi with p=0.7
    G_er_07 = nx.erdos_renyi_graph(graph_size, 0.7, seed=seed)
    if G_er_07.number_of_edges() == 0:
        G_er_07.add_edge(0, 1)
    graphs['Erdos-Renyi (p=0.7)'] = G_er_07

    return graphs

class GeneratedTestRunner:
    """Runner for generated graph tests."""

    def __init__(self, verbose=True, output_dir='results/generated_tests'):
        self.verbose = verbose
        self.output_dir = output_dir
        self.results = []
        # Create output directory on initialization
        self._ensure_dir(os.path.join(output_dir, 'dummy.csv'))

    def run_test_for_graph(self, G, graph_type, graph_size, trial_num):
        """
        Run test for a single graph.

        Args:
            G: NetworkX graph
            graph_type: Type of graph (e.g., 'Complete', 'Cycle')
            graph_size: Size of graph (number of nodes)

```

```

        trial_num: Trial number

Returns:
    GeneratedGraphResult object
    """
    result = GeneratedGraphResult(graph_type, graph_size, trial_num)
    result.num_edges = G.number_of_edges()

    try:
        # Run Brute Force
        start = time.time()
        result.bf_cut, _ = brute_force(G, debug=False)
        result.bf_time = time.time() - start

        # Run QASPA
        try:
            start = time.time()
            result.qaspa_cut, _, _ = run_qaspa_maxcut_local(
                G, debug=False, angle=np.pi/4, deep=1
            )
            result.qaspa_time = time.time() - start

            if result.bf_cut > 0:
                result.qaspa_ratio = result.qaspa_cut / result.bf_cut
                result.qaspa_success = result.qaspa_ratio >=
SUCCESS_THRESHOLD
        except Exception as e:
            if self.verbose:
                print(f"    QASPA ERROR: {str(e)}")
            result.qaspa_cut = 0
            result.qaspa_time = 0
            result.qaspa_ratio = 0

        # Run QAOA QRisp
        try:
            start = time.time()
            result.qaoa_cut, _ = qaoa_qrisp(G, debug=False)
            result.qaoa_time = time.time() - start

            if result.bf_cut > 0:
                result.qaoa_ratio = result.qaoa_cut / result.bf_cut
                result.qaoa_success = result.qaoa_ratio >=
SUCCESS_THRESHOLD
        except Exception as e:
            if self.verbose:
                print(f"    QAOA ERROR: {str(e)}")
            result.qaoa_cut = 0
            result.qaoa_time = 0
            result.qaoa_ratio = 0

        self.results.append(result)
        return result

    except Exception as e:
        if self.verbose:
            print(f"    Brute Force ERROR: {str(e)}")
        return None

def run_all_tests(self, num_trials=3):
    """
    Run tests for all graph sizes and types.

    Args:
        num_trials: Number of trials per graph type/size combination

```

```

"""
total_tests = len(GRAPH_SIZES) * 5 * num_trials # 5 graph types
current_test = 0

print("\n" + "="*80)
print("GENERATED GRAPH TESTING SUITE")
print("="*80)
print(f"Graph sizes: {GRAPH_SIZES[0]} to {GRAPH_SIZES[-1]}")
print(f"Graph types: Complete, Cycle, Erdős-Rényi (p=0.3, 0.5, 0.7)")
print(f"Trials per combination: {num_trials}")
print(f"Total tests: {total_tests}")
print(f"Success threshold: {SUCCESS_THRESHOLD}")
print("="*80 + "\n")

for graph_size in GRAPH_SIZES:
    print(f"\nTesting graphs with {graph_size} nodes...")

    for trial_num in range(num_trials):
        graphs = generate_graphs(graph_size, trial_num)

        for graph_type, G in graphs.items():
            current_test += 1

            # Progress indicator
            progress = f"[{current_test:3d}/{total_tests}]"
            type_info = f"{graph_type:20s} Size={graph_size:2d}"
            size_info = f"Edges={G.number_of_edges():3d}"
            print(f"    {progress} {type_info} {size_info}", end=" ...
", flush=True)

            # Run test
            result = self.run_test_for_graph(G, graph_type,
graph_size, trial_num)

            if result and result.bf_cut is not None:
                gaspa_info = f"QASPA: {result.gaspa_ratio:.3f}" if
result.gaspa_ratio else "QASPA: ERROR"
                qaoa_info = f"QAOA: {result.qaoa_ratio:.3f}" if
result.qaoa_ratio else "QAOA: ERROR"
                print(f"    {gaspa_info}, {qaoa_info}")
            else:
                print("FAILED")

def get_results_dataframe(self):
    """Get results as pandas DataFrame."""
    data = [r.to_dict() for r in self.results]
    return pd.DataFrame(data)

def get_summary_by_graph_type(self):
    """Get summary statistics by graph type."""
    df = self.get_results_dataframe()

    summaries = []
    for graph_type in df['Graph Type'].unique():
        subset = df[df['Graph Type'] == graph_type]

        gaspa_ratios = subset['QASPA Ratio'].dropna()
        qaoa_ratios = subset['QAOA Ratio'].dropna()

        gaspa_success_rate = subset['QASPA Success'].sum() / len(subset)
        * 100 if len(subset) > 0 else 0
        qaoa_success_rate = subset['QAOA Success'].sum() / len(subset) *
100 if len(subset) > 0 else 0

```

```

        summary = {
            'Graph Type': graph_type,
            'Tests': len(subset),
            'QASPA Mean Ratio': gaspa_ratios.mean() if len(gaspa_ratios)
> 0 else 0,
            'QASPA Success Rate %': gaspa_success_rate,
            'QAOA Mean Ratio': qaoa_ratios.mean() if len(qaoa_ratios) > 0
else 0,
            'QAOA Success Rate %': qaoa_success_rate,
            'Avg BF Time (s)': subset['BF Time (s)'].mean(),
            'Avg QASPA Time (s)': subset['QASPA Time (s)'].mean(),
            'Avg QAOA Time (s)': subset['QAOA Time (s)'].mean(),
        }
        summaries.append(summary)

    return pd.DataFrame(summaries)

def get_summary_by_size(self):
    """Get summary statistics by graph size."""
    df = self.get_results_dataframe()

    summaries = []
    for size in sorted(df['Size'].unique()):
        subset = df[df['Size'] == size]

        gaspa_ratios = subset['QASPA Ratio'].dropna()
        qaoa_ratios = subset['QAOA Ratio'].dropna()

        gaspa_success_rate = subset['QASPA Success'].sum() / len(subset)
* 100 if len(subset) > 0 else 0
        qaoa_success_rate = subset['QAOA Success'].sum() / len(subset) *
100 if len(subset) > 0 else 0

        summary = {
            'Size': size,
            'Tests': len(subset),
            'QASPA Mean Ratio': gaspa_ratios.mean() if len(gaspa_ratios)
> 0 else 0,
            'QASPA Success Rate %': gaspa_success_rate,
            'QAOA Mean Ratio': qaoa_ratios.mean() if len(qaoa_ratios) > 0
else 0,
            'QAOA Success Rate %': qaoa_success_rate,
            'Avg BF Time (s)': subset['BF Time (s)'].mean(),
            'Avg QASPA Time (s)': subset['QASPA Time (s)'].mean(),
            'Avg QAOA Time (s)': subset['QAOA Time (s)'].mean(),
        }
        summaries.append(summary)

    return pd.DataFrame(summaries)

def print_summary_by_type(self):
    """Print summary by graph type."""
    df = self.get_summary_by_graph_type()

    print("\n" + "="*140)
    print("SUMMARY BY GRAPH TYPE")
    print("="*140)
    print(df.to_string(index=False))
    print("="*140 + "\n")

def print_summary_by_size(self):
    """Print summary by graph size."""
    df = self.get_summary_by_size()

```

```

print("\n" + "="*140)
print("SUMMARY BY GRAPH SIZE")
print("="*140)
print(df.to_string(index=False))
print("="*140 + "\n")

def plot_success_rate_by_type(self):
    """Plot success rate by graph type."""
    df = self.get_summary_by_graph_type()

    fig, ax = plt.subplots(figsize=(12, 6))

    x = np.arange(len(df))
    width = 0.35

    ax.bar(x - width/2, df['QASPA Success Rate %'], width, label='QASPA',
alpha=0.8, color='#ff7f0e')
    ax.bar(x + width/2, df['QAOA Success Rate %'], width, label='QAOA
QRisp', alpha=0.8, color='#2ca02c')
    ax.axhline(y=100, color='r', linestyle='--', linewidth=2,
label='Perfect (100%)')

    ax.set_xlabel('Graph Type', fontsize=12, fontweight='bold')
    ax.set_ylabel('Success Rate (%)', fontsize=12, fontweight='bold')
    ax.set_title('Success Rate by Graph Type (ratio >= 0.9)',
fontsize=14, fontweight='bold')
    ax.set_xticks(x)
    ax.set_xticklabels(df['Graph Type'], rotation=45, ha='right')
    ax.legend(fontsize=11)
    ax.grid(axis='y', alpha=0.3)
    ax.set_ylim([0, 110])

    plt.tight_layout()
    save_path =
os.path.join(self.output_dir.replace('results/generated_tests',
'plots/generated_tests'),
              'success_rate_by_type.png')
    self._ensure_dir(save_path)
    plt.savefig(save_path, dpi=300, bbox_inches='tight')
    print(f"Saved: {save_path}")
    plt.close()

def plot_success_rate_by_size(self):
    """Plot success rate vs graph size."""
    df = self.get_summary_by_size()

    fig, ax = plt.subplots(figsize=(12, 6))

    ax.plot(df['Size'], df['QASPA Success Rate %'], 'o-', linewidth=2,
markersize=8,
            label='QASPA', color='#ff7f0e')
    ax.plot(df['Size'], df['QAOA Success Rate %'], 's-', linewidth=2,
markersize=8,
            label='QAOA QRisp', color='#2ca02c')
    ax.axhline(y=100, color='r', linestyle='--', linewidth=2,
label='Perfect (100%)')

    ax.set_xlabel('Graph Size (Number of Nodes)', fontsize=12,
fontweight='bold')
    ax.set_ylabel('Success Rate (%)', fontsize=12, fontweight='bold')
    ax.set_title('Success Rate vs Graph Size (ratio >= 0.9)',
fontsize=14, fontweight='bold')
    ax.legend(fontsize=11)

```

```

ax.grid(True, alpha=0.3)
ax.set_ylim([0, 110])

plt.tight_layout()
save_path =
os.path.join(self.output_dir.replace('results/generated_tests',
'plots/generated_tests'),
              'success_rate_by_size.png')
self._ensure_dir(save_path)
plt.savefig(save_path, dpi=300, bbox_inches='tight')
print(f"Saved: {save_path}")
plt.close()

def plot_approximation_ratio_by_type(self):
    """Plot mean approximation ratio by graph type."""
    df = self.get_summary_by_graph_type()

    fig, ax = plt.subplots(figsize=(12, 6))

    x = np.arange(len(df))
    width = 0.35

    ax.bar(x - width/2, df['QASPA Mean Ratio'], width, label='QASPA',
alpha=0.8, color='#ff7f0e')
    ax.bar(x + width/2, df['QAOA Mean Ratio'], width, label='QAOA QRisp',
alpha=0.8, color='#2ca02c')
    ax.axhline(y=1.0, color='r', linestyle='--', linewidth=2,
label='Optimal (1.0)')

    ax.set_xlabel('Graph Type', fontsize=12, fontweight='bold')
    ax.set_ylabel('Mean Approximation Ratio', fontsize=12,
fontweight='bold')
    ax.set_title('Mean Approximation Ratio by Graph Type', fontsize=14,
fontweight='bold')
    ax.set_xticks(x)
    ax.set_xticklabels(df['Graph Type'], rotation=45, ha='right')
    ax.legend(fontsize=11)
    ax.grid(axis='y', alpha=0.3)
    ax.set_ylim([0, 1.2])

    plt.tight_layout()
    save_path =
os.path.join(self.output_dir.replace('results/generated_tests',
'plots/generated_tests'),
              'approximation_ratio_by_type.png')
    self._ensure_dir(save_path)
    plt.savefig(save_path, dpi=300, bbox_inches='tight')
    print(f"Saved: {save_path}")
    plt.close()

def plot_approximation_ratio_by_size(self):
    """Plot mean approximation ratio vs graph size."""
    df = self.get_summary_by_size()

    fig, ax = plt.subplots(figsize=(12, 6))

    ax.plot(df['Size'], df['QASPA Mean Ratio'], 'o-', linewidth=2,
markersize=8,
          label='QASPA', color='#ff7f0e')
    ax.plot(df['Size'], df['QAOA Mean Ratio'], 's-', linewidth=2,
markersize=8,
          label='QAOA QRisp', color='#2ca02c')
    ax.axhline(y=1.0, color='r', linestyle='--', linewidth=2,
label='Optimal (1.0)')

```

```

        ax.set_xlabel('Graph Size (Number of Nodes)', fontsize=12,
fontweight='bold')
        ax.set_ylabel('Mean Approximation Ratio', fontsize=12,
fontweight='bold')
        ax.set_title('Mean Approximation Ratio vs Graph Size', fontsize=14,
fontweight='bold')
        ax.legend(fontsize=11)
        ax.grid(True, alpha=0.3)
        ax.set_ylim([0, 1.2])

    plt.tight_layout()
    save_path =
os.path.join(self.output_dir.replace('results/generated_tests',
'plots/generated_tests'),
                'approximation_ratio_by_size.png')
    self._ensure_dir(save_path)
    plt.savefig(save_path, dpi=300, bbox_inches='tight')
    print(f"Saved: {save_path}")
    plt.close()

def plot_execution_time_by_type(self):
    """Plot execution time by graph type."""
    df = self.get_summary_by_graph_type()

    fig, ax = plt.subplots(figsize=(12, 6))

    x = np.arange(len(df))
    width = 0.25

    ax.bar(x - width, df['Avg BF Time (s)'], width, label='Brute Force',
alpha=0.8, color='#1f77b4')
    ax.bar(x, df['Avg QASPA Time (s)'], width, label='QASPA', alpha=0.8,
color='#ff7f0e')
    ax.bar(x + width, df['Avg QAOA Time (s)'], width, label='QAOA QRisp',
alpha=0.8, color='#2ca02c')

    ax.set_xlabel('Graph Type', fontsize=12, fontweight='bold')
    ax.set_ylabel('Time (seconds)', fontsize=12, fontweight='bold')
    ax.set_title('Average Execution Time by Graph Type', fontsize=14,
fontweight='bold')
    ax.set_xticks(x)
    ax.set_xticklabels(df['Graph Type'], rotation=45, ha='right')
    ax.legend(fontsize=11)
    ax.grid(axis='y', alpha=0.3)

    plt.tight_layout()
    save_path =
os.path.join(self.output_dir.replace('results/generated_tests',
'plots/generated_tests'),
                'execution_time_by_type.png')
    self._ensure_dir(save_path)
    plt.savefig(save_path, dpi=300, bbox_inches='tight')
    print(f"Saved: {save_path}")
    plt.close()

def plot_execution_time_by_size(self):
    """Plot execution time vs graph size."""
    df = self.get_summary_by_size()

    fig, ax = plt.subplots(figsize=(12, 6))

    ax.plot(df['Size'], df['Avg BF Time (s)'], 'o-', linewidth=2,
markersize=8,

```

```

        label='Brute Force', color='#1f77b4')
    ax.plot(df['Size'], df['Avg QASPA Time (s)'], 's-', linewidth=2,
markersize=8,
        label='QASPA', color='#ff7f0e')
    ax.plot(df['Size'], df['Avg QAOA Time (s)'], '^-', linewidth=2,
markersize=8,
        label='QAOA QRisp', color='#2ca02c')

    ax.set_xlabel('Graph Size (Number of Nodes)', fontsize=12,
fontweight='bold')
    ax.set_ylabel('Time (seconds)', fontsize=12, fontweight='bold')
    ax.set_title('Average Execution Time vs Graph Size', fontsize=14,
fontweight='bold')
    ax.legend(fontsize=11)
    ax.grid(True, alpha=0.3)

    plt.tight_layout()
    save_path =
os.path.join(self.output_dir.replace('results/generated_tests',
'plots/generated_tests'),
              'execution_time_by_size.png')
    self._ensure_dir(save_path)
    plt.savefig(save_path, dpi=300, bbox_inches='tight')
    print(f"Saved: {save_path}")
    plt.close()

def generate_all_plots(self):
    """Generate all plots."""
    print("\n" + "="*80)
    print("GENERATING PLOTS...")
    print("="*80 + "\n")

    self.plot_success_rate_by_type()
    self.plot_success_rate_by_size()
    self.plot_approximation_ratio_by_type()
    self.plot_approximation_ratio_by_size()
    self.plot_execution_time_by_type()
    self.plot_execution_time_by_size()

    print("\n" + "="*80)

def save_results_to_csv(self):
    """Save results to CSV files."""
    print("\nSaving results to CSV...\n")

    # Ensure output directory exists
    self._ensure_dir(os.path.join(self.output_dir, 'dummy.csv'))

    # Detailed results
    df = self.get_results_dataframe()
    detailed_path = os.path.join(self.output_dir, 'detailed_results.csv')
    df.to_csv(detailed_path, index=False)
    print(f"Saved: {detailed_path}")

    # Summary by type
    df_by_type = self.get_summary_by_graph_type()
    summary_type_path = os.path.join(self.output_dir,
'summary_by_type.csv')
    df_by_type.to_csv(summary_type_path, index=False)
    print(f"Saved: {summary_type_path}")

    # Summary by size
    df_by_size = self.get_summary_by_size()

```

```

        summary_size_path = os.path.join(self.output_dir,
'summary_by_size.csv')
        df_by_size.to_csv(summary_size_path, index=False)
        print(f"Saved: {summary_size_path}\n")

    @staticmethod
    def _ensure_dir(filepath):
        """Ensure directory exists for filepath."""
        directory = os.path.dirname(filepath)
        if directory and not os.path.exists(directory):
            os.makedirs(directory, exist_ok=True)

def main():
    """Main test execution."""
    runner = GeneratedTestRunner(verbose=True,
output_dir='results/generated_tests')

    # Run all tests (3 trials per combination)
    runner.run_all_tests(num_trials=3)

    # Print summaries
    runner.print_summary_by_type()
    runner.print_summary_by_size()

    # Generate plots
    runner.generate_all_plots()

    # Save results
    runner.save_results_to_csv()

    print("="*80)
    print("GENERATED GRAPH TESTING COMPLETED!")
    print("="*80 + "\n")

if __name__ == "__main__":
    main()

```

angle_optimization.py

```

import os
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
from max_cut_gaspa import run_gaspa_maxcut_local
from brute_force import brute_force

# Create output directory
os.makedirs('angle_results', exist_ok=True)

# Test angles: pi/2, pi/3, pi/4
angles = [np.pi/2, np.pi/3, np.pi/4]

# Number of trials per angle
num_trials = 10

# Number of nodes in random graphs
graph_sizes = list(range(3, 15))

# Success threshold: 90% (ratio >= 0.9)
SUCCESS_THRESHOLD = 0.9

base_seed = 42

```

```

# Define different graph types
def generate_graphs(graph_size, trial_num):
    """Generate different types of graphs for testing"""
    graphs = {}
    # Use seed 42 for reproducibility
    # 1. Complete graph
    G_complete = nx.complete_graph(graph_size)
    graphs['Complete'] = G_complete

    # 2. Cycle graph
    G_cycle = nx.cycle_graph(graph_size)
    graphs['Cycle'] = G_cycle

    # 3. Erdős-Rényi with p=0.3
    G_er_03 = nx.erdos_renyi_graph(graph_size, 0.3, seed=base_seed)
    if G_er_03.number_of_edges() == 0:
        G_er_03.add_edge(0, 1)
    graphs['Erdos-Renyi (p=0.3)'] = G_er_03

    # 4. Erdős-Rényi with p=0.5
    G_er_05 = nx.erdos_renyi_graph(graph_size, 0.5, seed=base_seed)
    if G_er_05.number_of_edges() == 0:
        G_er_05.add_edge(0, 1)
    graphs['Erdos-Renyi (p=0.5)'] = G_er_05

    # 5. Erdős-Rényi with p=0.7
    G_er_07 = nx.erdos_renyi_graph(graph_size, 0.7, seed=base_seed)
    if G_er_07.number_of_edges() == 0:
        G_er_07.add_edge(0, 1)
    graphs['Erdos-Renyi (p=0.7)'] = G_er_07

    return graphs

# Store results for all graph types
all_results = {}

print("Starting angle optimization analysis...")
print(f"Angles to test: {[f'{angle:.4f}' for angle in angles]}")
print(f"Trials per angle: {num_trials}")
print(f"Graph sizes: {graph_sizes}\n")

# Test for each graph type
for angle_idx, angle in enumerate(angles):
    print(f"\n{'='*60}")
    print(f"Testing angle: {angle:.4f} rad ({np.degrees(angle):.2f}°)")
    print(f"{'='*60}")

    # Initialize results for this angle
    angle_key = f'{angle:.4f}'
    if angle_key not in all_results:
        all_results[angle_key] = {}

    for graph_size in graph_sizes:
        for trial in range(num_trials):
            # Generate different graph types
            graphs = generate_graphs(
                graph_size, angle_idx*1000 + graph_size*100 + trial)

            for graph_type, G in graphs.items():
                # Initialize storage for this graph type if needed
                if graph_type not in all_results[angle_key]:
                    all_results[angle_key][graph_type] = {}

```

```

        'correct': 0,
        'total': 0,
        'ratios': [],
        'success_rate': 0,
        'avg_ratio': 0
    }

    # Run brute force to get optimal cut
    try:
        brute_force_cut, _ = brute_force(G)
    except Exception as e:
        print(
            f" Error: brute force failed for {graph_type} (size
{graph_size}): {str(e)}")
        continue

    # Run QASPA with the current angle
    try:
        gaspa_cut, _, _ = run_gaspa_maxcut_local(
            G, debug=False, angle=angle)

        all_results[angle_key][graph_type]['total'] += 1

        # Calculate approximation ratio
        approx_ratio = gaspa_cut / brute_force_cut if
brute_force_cut > 0 else 0
        all_results[angle_key][graph_type]['ratios'].append(
            approx_ratio)

        # Check if result meets 90% success threshold
        if approx_ratio >= SUCCESS_THRESHOLD:
            all_results[angle_key][graph_type]['correct'] += 1

    except Exception as e:
        print(
            f" Error in {graph_type} (size {graph_size}, trial
{trial+1}): {str(e)}")
        continue

    # Print summary for this angle
    print(f"\nSummary for angle {angle:.4f} rad:")
    print("-" * 60)
    for graph_type in sorted(all_results[angle_key].keys()):
        data = all_results[angle_key][graph_type]
        if data['total'] > 0:
            data['success_rate'] = (data['correct'] / data['total']) * 100
            data['avg_ratio'] = np.mean(
                data['ratios']) if data['ratios'] else 0

        print(
            f"{graph_type:20s}: {data['correct']:3d}/{data['total']:3d}
({data['success_rate']:5.1f}%) | Avg ratio: {data['avg_ratio']:.4f}")

    # Find optimal angles for each graph type
    print(f"\n\n{'='*60}")
    print("OPTIMAL ANGLES FOUND")
    print(f"{'='*60}\n")

    # Organize results by graph type
    results_by_type = {}
    for angle_key in all_results:
        for graph_type in all_results[angle_key]:
            if graph_type not in results_by_type:
                results_by_type[graph_type] = {
```

```

        'angles': [],
        'success_rates': [],
        'avg_ratios': []
    }
    results_by_type[graph_type]['angles'].append(float(angle_key))
    results_by_type[graph_type]['success_rates'].append(
        all_results[angle_key][graph_type]['success_rate'])
    results_by_type[graph_type]['avg_ratios'].append(
        all_results[angle_key][graph_type]['avg_ratio'])

# Find and print optimal angle for each graph type
optimal_info = {}
for graph_type in sorted(results_by_type.keys()):
    data = results_by_type[graph_type]
    if data['success_rates']:
        optimal_idx = np.argmax(data['success_rates'])
        optimal_angle = data['angles'][optimal_idx]
        optimal_success = data['success_rates'][optimal_idx]
        optimal_info[graph_type] = {
            'angle': optimal_angle,
            'success_rate': optimal_success,
            'avg_ratio': data['avg_ratios'][optimal_idx]
        }

    print(f"{graph_type}:")
    print(
        f"    Optimal angle: {optimal_angle:.4f} rad
({np.degrees(optimal_angle):.2f}°)")
    print(f"    Success rate: {optimal_success:.2f}%")
    print(
        f"    Average approximation ratio:
{data['avg_ratios'][optimal_idx]:.4f}\n")

# =====
# FIND OVERALL OPTIMAL ANGLE FOR ALL GRAPH TYPES
# =====
print(f"\n{'='*60}")
print("FINDING OVERALL OPTIMAL ANGLE")
print(f"{'='*60}\n")

# Calculate average success rate for each angle across all graph types
angle_avg_success = {}
angle_avg_ratio = {}
for angle_key in all_results:
    angle_float = float(angle_key)
    total_success = 0
    total_ratio = 0
    count = 0

    for graph_type in all_results[angle_key]:
        data = all_results[angle_key][graph_type]
        if data['total'] > 0:
            total_success += data['success_rate']
            total_ratio += data['avg_ratio']
            count += 1

    if count > 0:
        angle_avg_success[angle_float] = total_success / count
        angle_avg_ratio[angle_float] = total_ratio / count

# Find the angle with highest average success rate
best_angle = max(angle_avg_success, key=angle_avg_success.get)
best_angle_deg = np.degrees(best_angle)
best_avg_success = angle_avg_success[best_angle]

```

```

best_avg_ratio = angle_avg_ratio[best_angle]

print(f"BEST OVERALL ANGLE: {best_angle:.4f} rad ({best_angle_deg:.2f}°)")
print(f" Average success rate across all graph types:
{best_avg_success:.2f}%")
print(f" Average approximation ratio: {best_avg_ratio:.4f}\n")

print(f"Results for ALL graph types with angle {best_angle_deg:.2f}°:")
print("-" * 60)
best_angle_key = f'{best_angle:.4f}'
for graph_type in sorted(all_results[best_angle_key].keys()):
    data = all_results[best_angle_key][graph_type]
    print(f"{graph_type:25s}: {data['correct']:3d}/{data['total']:3d}
({data['success_rate']:5.1f}%) | Avg ratio: {data['avg_ratio']:.4f}")

# Save results to file
with open('angle_results/optimization_results.txt', 'w') as f:
    f.write("ANGLE OPTIMIZATION RESULTS\n")
    f.write("="*60 + "\n\n")

    f.write("OVERALL OPTIMAL ANGLE:\n")
    f.write(f" Angle: {best_angle:.4f} rad ({best_angle_deg:.2f}°)\n")
    f.write(f" Average success rate: {best_avg_success:.2f}%\n")
    f.write(f" Average approximation ratio: {best_avg_ratio:.4f}\n\n")

    f.write("="*60 + "\n")
    f.write("RESULTS FOR ALL GRAPH TYPES WITH OPTIMAL ANGLE\n")
    f.write("="*60 + "\n\n")

    for graph_type in sorted(all_results[best_angle_key].keys()):
        data = all_results[best_angle_key][graph_type]
        f.write(f"{graph_type}:\n")
        f.write(f" Success: {data['correct']}/{data['total']}
({data['success_rate']:.2f}%) \n")
        f.write(f" Average approximation ratio:
{data['avg_ratio']:.4f}\n\n")

    f.write("\n" + "="*60 + "\nDETAILED RESULTS FOR ALL ANGLES AND GRAPH
TYPES\n")
    f.write("="*60 + "\n\n")

    for angle_key in sorted(all_results.keys()):
        angle_float = float(angle_key)
        marker = " ← BEST" if angle_float == best_angle else ""
        f.write(
            f"Angle: {angle_key} rad
({np.degrees(angle_float):.2f}°){marker}\n")
        f.write("-" * 60 + "\n")
        for graph_type in sorted(all_results[angle_key].keys()):
            data = all_results[angle_key][graph_type]
            f.write(f" {graph_type}: {data['correct']}/{data['total']} ")
            f.write(
                f"({data['success_rate']:.2f}%) | Avg ratio:
{data['avg_ratio']:.4f}\n")
            f.write("\n")

# Create final visualization: Results for all graph types with optimal angle
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

graph_types = sorted(all_results[best_angle_key].keys())
success_rates = [all_results[best_angle_key][gt]['success_rate'] for gt in
graph_types]
avg_ratios = [all_results[best_angle_key][gt]['avg_ratio'] for gt in
graph_types]

```

```

# Shorten graph type names for better display
short_names = []
for gt in graph_types:
    if gt == 'Complete':
        short_names.append('Complete')
    elif gt == 'Cycle':
        short_names.append('Cycle')
    else:
        p = gt.split('=')[1].strip(' ')
        short_names.append(f'ER(p={p})')

colors = plt.cm.Set3(np.linspace(0, 1, len(graph_types)))
x_pos = np.arange(len(graph_types))

# Success rate plot
bars1 = ax1.bar(x_pos, success_rates, color=colors, alpha=0.8,
edgecolor='black', linewidth=1.5)
ax1.axhline(y=90, color='green', linestyle='--', linewidth=2, label='90%
threshold')
ax1.set_xlabel('Graph Type', fontsize=12, fontweight='bold')
ax1.set_ylabel('Success Rate (%)', fontsize=12, fontweight='bold')
ax1.set_title(f'Success Rate for All Graph Types\nOptimal Angle:
{best_angle_deg:.2f}° ( $\pi/{best\_angle:.4f}$ )',
              fontsize=13, fontweight='bold')
ax1.set_xticks(x_pos)
ax1.set_xticklabels(short_names, rotation=45, ha='right')
ax1.set_ylim([0, 105])
ax1.grid(True, alpha=0.3, axis='y')
ax1.legend(fontsize=11)

# Add value labels on bars
for i, (bar, val) in enumerate(zip(bars1, success_rates)):
    ax1.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 1.5,
             f'{val:.1f}%', ha='center', va='bottom', fontsize=10,
             fontweight='bold')

# Approximation ratio plot
bars2 = ax2.bar(x_pos, avg_ratios, color=colors, alpha=0.8,
edgecolor='black', linewidth=1.5)
ax2.axhline(y=0.9, color='green', linestyle='--', linewidth=2, label='0.9
threshold (90%)')
ax2.axhline(y=1.0, color='red', linestyle='--', linewidth=2, label='Optimal
(1.0)')
ax2.set_xlabel('Graph Type', fontsize=12, fontweight='bold')
ax2.set_ylabel('Average Approximation Ratio', fontsize=12, fontweight='bold')
ax2.set_title(f'Approximation Ratio for All Graph Types\nOptimal Angle:
{best_angle_deg:.2f}° ( $\pi/{best\_angle:.4f}$ )',
              fontsize=13, fontweight='bold')
ax2.set_xticks(x_pos)
ax2.set_xticklabels(short_names, rotation=45, ha='right')
ax2.set_ylim([0.75, 1.05])
ax2.grid(True, alpha=0.3, axis='y')
ax2.legend(fontsize=11)

# Add value labels on bars
for i, (bar, val) in enumerate(zip(bars2, avg_ratios)):
    ax2.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.01,
             f'{val:.3f}', ha='center', va='bottom', fontsize=10,
             fontweight='bold')

plt.tight_layout()
plt.savefig('angle_results/final_results.png', dpi=100, bbox_inches='tight')
print("\nFinal results plot saved to: angle_results/final_results.png")

```

```

# Create comparison plot for all angles
fig, ax = plt.subplots(figsize=(12, 6))

unique_angles = sorted(list(set([float(k) for k in all_results.keys()])))
angles_deg = [np.degrees(a) for a in unique_angles]

# Calculate average success and ratio for each angle
avg_success_per_angle = []
avg_ratio_per_angle = []
for angle in unique_angles:
    angle_key = f'{angle:.4f}'
    total_success = 0
    total_ratio = 0
    count = 0
    for graph_type in all_results[angle_key]:
        if all_results[angle_key][graph_type]['total'] > 0:
            total_success +=
all_results[angle_key][graph_type]['success_rate']
            total_ratio += all_results[angle_key][graph_type]['avg_ratio']
            count += 1
    if count > 0:
        avg_success_per_angle.append(total_success / count)
        avg_ratio_per_angle.append(total_ratio / count)

ax.plot(angles_deg, avg_success_per_angle, 'o-', linewidth=3, markersize=10,
        label='Avg Success Rate', color='blue')
ax.scatter([best_angle_deg], [best_avg_success], color='red', s=300,
           zorder=5,
           marker='*', label='Optimal Angle', edgecolors='darkred',
           linewidths=2)
ax.axhline(y=90, color='green', linestyle='--', linewidth=2, alpha=0.7,
           label='90% threshold')
ax.set_xlabel('Angle (degrees)', fontsize=12, fontweight='bold')
ax.set_ylabel('Average Success Rate (%)', fontsize=12, fontweight='bold')
ax.set_title('Average Success Rate Across All Graph Types vs Angle',
             fontsize=13, fontweight='bold')
ax.grid(True, alpha=0.3)
ax.legend(fontsize=11)
ax.set_ylim([0, 105])

# Add angle labels on x-axis
ax.set_xticks(angles_deg)
ax.set_xticklabels([f'{d:.0f}' for d in angles_deg])

plt.tight_layout()
plt.savefig('angle_results/angle_comparison.png', dpi=100,
           bbox_inches='tight')
print("Angle comparison plot saved to: angle_results/angle_comparison.png")

# Create comprehensive final diagram with all graph types
print("\nCreating comprehensive final diagram...")
fig, axes = plt.subplots(2, 1, figsize=(14, 10))

# Get unique angles
unique_angles = sorted(list(set([float(k) for k in all_results.keys()])))
angles_deg = [np.degrees(a) for a in unique_angles]

# Define colors for each graph type
graph_types_sorted =
sorted(list(all_results[list(all_results.keys())[0]].keys()))
colors_map = {
    'Complete': '#FF6B6B',
    'Cycle': '#4ECDC4',

```

```

'Erdos-Renyi (p=0.3)': '#45B7D1',
'Erdos-Renyi (p=0.5)': '#FFA07A',
'Erdos-Renyi (p=0.7)': '#98D8C8'
}

# Plot 1: Success Rate for all graph types across all angles
ax1 = axes[0]
for graph_type in graph_types_sorted:
    success_rates = []
    for angle in unique_angles:
        angle_key = f'{angle:.4f}'

    success_rates.append(all_results[angle_key][graph_type]['success_rate'])

    color = colors_map.get(graph_type, '#999999')
    ax1.plot(angles_deg, success_rates, 'o-', linewidth=2.5, markersize=8,
            label=graph_type, color=color, alpha=0.8)

# Mark optimal angle
best_angle_idx = [a for a in unique_angles].index(best_angle)
for i, graph_type in enumerate(graph_types_sorted):
    angle_key = f'{best_angle:.4f}'
    success_rate = all_results[angle_key][graph_type]['success_rate']
    ax1.scatter(best_angle_deg, success_rate, s=250, color='green',
                zorder=10,
                marker='o', edgecolors='darkgreen', linewidths=2.5)

ax1.axhline(y=90, color='red', linestyle='--', linewidth=2, alpha=0.6,
            label='90% Threshold')
ax1.set_xlabel('Angle (degrees)', fontsize=12, fontweight='bold')
ax1.set_ylabel('Success Rate (%)', fontsize=12, fontweight='bold')
ax1.set_title('Success Rate for All Graph Types Across All Angles',
            fontsize=13, fontweight='bold')
ax1.grid(True, alpha=0.3)
ax1.legend(fontsize=10, loc='best')
ax1.set_ylim([-5, 105])
ax1.set_xticks(angles_deg)
ax1.set_xticklabels([f'{d:.0f}' for d in angles_deg])

# Plot 2: Approximation Ratio for all graph types across all angles
ax2 = axes[1]
for graph_type in graph_types_sorted:
    avg_ratios = []
    for angle in unique_angles:
        angle_key = f'{angle:.4f}'
        avg_ratios.append(all_results[angle_key][graph_type]['avg_ratio'])

    color = colors_map.get(graph_type, '#999999')
    ax2.plot(angles_deg, avg_ratios, 's-', linewidth=2.5, markersize=8,
            label=graph_type, color=color, alpha=0.8)

# Mark optimal angle
for i, graph_type in enumerate(graph_types_sorted):
    angle_key = f'{best_angle:.4f}'
    avg_ratio = all_results[angle_key][graph_type]['avg_ratio']
    ax2.scatter(best_angle_deg, avg_ratio, s=250, color='green', zorder=10,
                marker='o', edgecolors='darkgreen', linewidths=2.5)

ax2.axhline(y=0.9, color='red', linestyle='--', linewidth=2, alpha=0.6,
            label='0.9 Threshold (90%)')
ax2.axhline(y=1.0, color='orange', linestyle='--', linewidth=2, alpha=0.6,
            label='Optimal (1.0)')
ax2.set_xlabel('Angle (degrees)', fontsize=12, fontweight='bold')
ax2.set_ylabel('Average Approximation Ratio', fontsize=12, fontweight='bold')

```

```

ax2.set_title('Approximation Ratio for All Graph Types Across All Angles',
              fontsize=13, fontweight='bold')
ax2.grid(True, alpha=0.3)
ax2.legend(fontsize=10, loc='best')
ax2.set_ylim([0.75, 1.05])
ax2.set_xticks(angles_deg)
ax2.set_xticklabels([f'{d:.0f}°' for d in angles_deg])

plt.tight_layout()
plt.savefig('angle_results/comprehensive_results.png', dpi=100,
          bbox_inches='tight')
print("Comprehensive results plot saved to:
angle_results/comprehensive_results.png")
print("\n" + "="*60)
print("OPTIMIZATION COMPLETED")
print("="*60)
print(f"\nOPTIMAL ANGLE FOR ALL GRAPH TYPES: {best_angle:.4f} rad
({best_angle_deg:.2f}°)")
print(f"Average success rate: {best_avg_success:.2f}%")
print(f"Average approximation ratio: {best_avg_ratio:.4f}")
print(f"\nOutput files saved to: angle_results/")
print(f" - comprehensive_results.png: All results with lines for each graph
type")
print(f" - final_results.png: Results for all graph types with optimal
angle")
print(f" - angle_comparison.png: Success rate comparison across angles")
print(f" - optimization_results.txt: Detailed results")
print("="*60)

```

Список публікацій здобувача за темою дисертації

Сапожника Дмитра Олеговича

«Методи та алгоритми для розв’язання задачі максимального розрізу за допомогою квантово-гібридних алгоритмів»
зі спеціальності 121 «Інженерія програмного забезпечення»

**Статті у виданнях, включених до переліку наукових фахових видань
України категорії «Б»:**

1. Сапожник Д. О. Процес перетворення високорівневих квантових вентилів у базові для виконання на реальному квантовому комп’ютері. *Вісник Херсонського національного технічного університету*. 2025. Т. 2, № 2(93). С. 327–332. DOI: <https://doi.org/10.35546/kntu2078-4481.2025.2.2.40>.

2. Сапожник Д. О. Аналіз можливостей мови програмування Q# шляхом реалізації програми для генерації випадкових чисел. *Технічна інженерія*. 2024. № 1(93). С. 240–245. DOI: [https://doi.org/10.26642/ten-2024-1\(93\)-240-245](https://doi.org/10.26642/ten-2024-1(93)-240-245).

3. Сапожник Д. О. Розв’язання задачі максимального розрізу графа за допомогою квантово-гібридного амплітудно-стохастичного алгоритму. *Вісник Національного технічного університету «ХПІ». Серія: Системний аналіз, управління та інформаційні технології*. 2025. № 2 (14). С. 96–101. DOI: <https://doi.org/10.20998/2079-0023.2025.02.13>

4. Сапожник Д. О. Методи та напрями застосування квантових комп’ютерів у криптографії та логістиці. *Вчені записки ТНУ імені В.І. Вернадського. Серія: Технічні науки*. 2025. Т. 2, № 5. С. 260–265. DOI: <https://doi.org/10.32782/2663-5941/2025.5.2/34>

Публікації в зарубіжних періодичних фахових виданнях:

5. Sapozhnyk D. Quantum Approximate Optimization Algorithm for the Max-Cut Problem: JavaScript Programming Language Implementation. *Proceedings of International Conference on Applied Innovation in IT*. 2024. Vol. 12, iss. 2. Pp. 27–

34. DOI: <https://doi.org/10.25673/118110>. (включений до міжнародної наукометричної бази Scopus).

Публікації у матеріалах наукових конференцій:

6. Сапожник Д. О. Квантове програмування мовою Q#: перспективи та застосування. *Тези XIV Міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології – 2024»*. Житомир. 28–29 берез. 2024. С. 104–105.

7. Сапожник Д. О. Квантовий гейт Адамара, реалізація у мові програмування Q#. *Тези Всеукраїнської науково-практичної on-line конференції здобувачів вищої освіти і молодих учених, присвяченої Дню науки*. Житомир. 13–17 трав. 2024. С. 50.

8. Сапожник Д. О. Перспективи розв'язання пр-повних задач за допомогою квантового програмування. *Тези XIV міжнародної наукової конференції студентів та молодих вчених «Сучасні інформаційні технології 2024»*, Одеса. 24 трав. 2024. С. 98–100.

9. Сапожник Д. О. Програмні засоби поширення публічної бібліотеки для роботи з квантовими обчисленнями у мові програмування javascript. *Проблеми інформатики та моделювання (ПІМ-2024) тези двадцять четвертої міжнародної науково-технічної конференції*. Харків. 20–23 верес. 2024. С. 123–124.

10. Сапожник Д. О., Морозов А. В. Економічні та промислові застосування квантових комп'ютерів. *Сучасні інформаційні системи та технології: матеріали VII Всеукр. наук. практ. інтернет-конф. за тематикою «Сучасні комп'ютерні системи та мережі в управлінні» (29 листопада 2024 р., м. Херсон, м. Хмельницький) / за ред. А. А. Григорової*. Херсон: Книжкове видавництво ФОП Вишемирський В. 2024. С.113-116.

11. Сапожник Д. О. QUBO матриці для розв'язання пр-повних задач на квантовому комп'ютері. *Тези доповідей VII Всеукраїнської науково-технічної конференції Комп'ютерні технології: інновації, проблеми, рішення*. Житомир. 2–3 груд. 2024. С. 242–243.

Відомості про апробацію результатів дисертації у вигляді:

Масові науково-практичні заходи міжнародного рівня:

1. Сапожник Д. О. Квантове програмування мовою Q#: перспективи та застосування. Тези XIV Міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології – 2024», Житомир, 28–29 берез. 2024. С. 104–105.

2. Сапожник Д. О. Перспективи розв'язання пр-повних задач за допомогою квантового програмування. Тези XIV міжнародної наукової конференції студентів та молодих вчених «Сучасні інформаційні технології 2024», Одеса, 24 трав. 2024. С. 98–100.

3. Сапожник Д. О. Програмні засоби поширення публічної бібліотеки для роботи з квантовими обчисленнями у мові програмування javascript. *Проблеми інформатики та моделювання (ПІМ-2024)* тези двадцять четвертої міжнародної науково-технічної конференції, Харків, 20–23 верес. 2024. С. 123–124.

Масові науково-практичні заходи всеукраїнського рівня:

4. Сапожник Д. О. Квантовий гейт Адамара, реалізація у мові програмування Q#. Тези *Всеукраїнської науково-практичної on-line конференції здобувачів вищої освіти і молодих учених, присвяченої Дню науки*, Житомир, 13–17 трав. 2024. С. 50.

5. Сапожник Д. О., Морозов А. В. Економічні та промислові застосування квантових комп'ютерів. *Сучасні інформаційні системи та технології*: матеріали VII Всеукр. наук. практ. інтернет-конф. за тематикою «Сучасні комп'ютерні системи та мережі в управлінні» (29 листопада 2024 р., м. Херсон, м. Хмельницький) / за ред. А. А. Григорової. Херсон: Книжкове видавництво ФОП Вишемирський В. 2024. С.113-116.

6. Сапожник Д. О. QUBO матриці для розв'язання пр-повних задач на квантовому комп'ютері. Тези доповідей VII Всеукраїнської науково-технічної

конференції Комп'ютерні технології: інновації, проблеми, рішення, 2–3 груд. 2024. С. 242–243.

Масові науково-практичні заходи регіонального рівня:

7. Засідання міжкафедрального науково-методологічного семінару для молодих учених Державного університету «Житомирська політехніка» (Україна, м. Житомир, Державний університет «Житомирська політехніка», 21 січня 2026 р.). Форма участі – виступ з презентацією за результатами дисертаційного дослідження на тему: «Методи та алгоритми для розв'язання задачі максимального розрізу за допомогою квантово-гібридних алгоритмів».

Довідки про впровадження результатів дисертації
Сапожника Дмитра Олеговича
«Методи та алгоритми для розв'язання задачі максимального розрізу за
допомогою квантово-гібридних алгоритмів»
зі спеціальності 121 «Інженерія програмного забезпечення»
(СКАНОВАНІ КОПІЇ)

N-ix

Довідка
про впровадження результатів наукового дослідження
САПОЖНИКА Дмитра Олеговича
на тему
«МЕТОДИ ТА АЛГОРИТМИ ДЛЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ
МАКСИМАЛЬНОГО РОЗРІЗУ ЗА ДОПОМОГОЮ КВАНТОВО-ГІБРИДНИХ
АЛГОРИТМІВ»
на здобуття наукового ступеня доктора філософії
за спеціальністю 121 Інженерія програмного забезпечення
у галузі знань 12 Інформаційні технології

Результати дисертаційного дослідження САПОЖНИКА Дмитра Олеговича були апробовані та використані у діяльності Товариства з обмеженою відповідальністю «Н-ІКС ДЕЛІВЕРІ» в межах розробки експериментального програмного забезпечення для оптимізації логістичних процесів.

У межах дослідницьких та експериментальних робіт було застосовано квантово-гібридний алгоритм QASPA (Quantum Approximate Shift-Phase Algorithm) для дослідження підходів до оптимального розподілу логістичних маршрутів, кластеризації пунктів доставки та балансування транспортних ресурсів.

Алгоритм QASPA було інтегровано з класичними методами попередньої обробки даних та аналітичними модулями логістичної інформаційної системи, що дозволило підвищити ефективність прийняття рішень у задачах маршрутизації та планування доставки в умовах обмежених ресурсів та змінної динаміки попиту.

Результати експериментального впровадження засвідчили доцільність використання квантово-гібридних алгоритмів для задач комбінаторної оптимізації в логістиці, а також підтвердили перспективність застосування розроблених методів, моделей та алгоритмів у дисертаційному дослідженні САПОЖНИКА Д.О.

Директор

ТОВ «Н-ІКС ДЕЛІВЕРІ»



МИРОНЕНКО В. В.

Довідка
про впровадження результатів наукового дослідження
аспіранта кафедри інженерії програмного забезпечення
Державного університету «Житомирська політехніка»
САПОЖНИКА Дмитра Олеговича
на тему
«МЕТОДИ ТА АЛГОРИТМИ ДЛЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ
МАКСИМАЛЬНОГО РОЗРІЗУ ЗА ДОПОМОГОЮ КВАНТОВО-
ГІБРИДНИХ АЛГОРИТМІВ»

Дана довідка підтверджує, що результати дисертаційного дослідження САПОЖНИКА Дмитра Олеговича, проходили апробацію та були впроваджені у діяльність ТОВ «Альянс Діджитал» у вигляді застосування квантово-гібридного алгоритму QASPA (Quantum Approximate Shift-Phase Algorithm) в одному з напрямків розробки експериментального програмного додатка для планування та оптимізації інвестиційного портфеля.

У процесі впровадження було використано запропоновані у дисертаційній роботі методи квантово-гібридної оптимізації, які дозволили формалізувати задачу вибору оптимальної структури інвестиційного портфеля у вигляді задачі максимального розрізу.

Результати практичного використання алгоритму засвідчили його ефективність, масштабованість та перспективність застосування для розв'язання задач комбінаторної оптимізації у фінансових інформаційних системах, а також підтвердили доцільність використання квантово-гібридних підходів у прикладних програмних продуктах.

Отримані під час впровадження результати можуть бути використані у подальших науково-дослідних та прикладних розробках ТОВ «Альянс Діджитал», що свідчить про актуальність та цінність результатів дисертаційного дослідження САПОЖНИКА Д.О.

Директор



Марчук Б.О.

**ТОВАРИСТВО З ОБМЕЖЕНОЮ ВІДПОВІДАЛЬНІСТЮ
«УКРАЇНСЬКИЙ ЦЕНТР ДУАЛЬНОЇ ОСВІТИ»**

01030, м. Київ, вул. Богдана Хмельницького, 51 Б, ЄДРПОУ 42637285

м. Житомир

02.02.2026

Довідка

про впровадження результатів наукового дослідження

САПОЖНИКА Дмитра Олеговича

на тему

**«МЕТОДИ ТА АЛГОРИТМИ ДЛЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ
МАКСИМАЛЬНОГО РОЗРІЗУ ЗА ДОПОМОГОЮ
КВАНТОВО-ГІБРИДНИХ АЛГОРИТМІВ»**

на здобуття наукового ступеня доктора філософії
за спеціальністю 121 Інженерія програмного забезпечення
у галузі знань 12 Інформаційні технології

Засвідчую, що результати дисертаційного дослідження САПОЖНИКА Дмитра Олеговича були апробовані та використані у межах R&D-діяльності ТОВ «УКРАЇНСЬКИЙ ЦЕНТР ДУАЛЬНОЇ ОСВІТИ» під час виконання дослідницьких та експериментальних робіт, спрямованих на дослідження можливостей застосування квантово-гібридних алгоритмів у програмній інженерії.

У процесі R&D-досліджень було використано розроблений у дисертаційній роботі квантово-гібридний алгоритм QASPA (Quantum Approximate Shift-Phase Algorithm) для створення та аналізу експериментальних програмних прототипів, орієнтованих на розв'язання задач комбінаторної оптимізації, представлених у вигляді графових моделей. Алгоритм застосовувався для дослідження підходів до оптимального поділу елементів програмних систем, кластеризації та розподілу ресурсів.

Результати проведених дослідницьких та експериментальних робіт підтвердили наукову обґрунтованість запропонованих у дисертації методів, а також їх актуальність та перспективність для використання при створенні експериментальних програмних продуктів, proof-of-concept рішень та подальших прикладних досліджень у сфері інженерії програмного забезпечення. Це підтверджує важливість результатів отриманих у дисертаційному дослідженні САПОЖНИКА Д. О., а запропоновані моделі та алгоритм є актуальними та мають практичну цінність.

Довідку видано для подання за місцем вимоги.

Директор

Єршов М.В.

 ДОКУМЕНТ

Заголовок

Дисертація Сапожник

ARTON

Дмитро Сапожник

Науковий керівник / Експерт

Zhytomyr Polytechnic

ІД документу

333547188

Назва організації

References

State University "Zhytomyr Polytechnic"

підрозділ

State University "Zhytomyr Polytechnic"

 3BIT

Дата звіту

4/7/2026

Дата редагування

Да

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



25

Довжина фрази для коефіцієнта подібності 2



29382

Кількість слів

228071

Кількість символів

 Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про **МОЖЛИВІ** маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам відходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		3
Інтервали		0
Мікропробіли		231
Білі знаки		0
Парафрази (SmartMarks)		14

Джерела

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз		Колір тексту
#	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://ztu.edu.ua/ua/science/files/onelife/vysnovok_Naumchuk.pdf	15 (0.05 %)

2	http://referatu.net.ua/referats/7569/173568	11 (0.04 %)
3	https://iq.vntu.edu.ua/repository/getfile.php/9117.pdf	11 (0.04 %)
4	http://repository.ub.ac.id/125360/6/12_Lampiran.pdf	10 (0.03 %)
5	https://ztu.edu.ua/ua/science/files/onelife/vysnovok_Naumchuk.pdf	10 (0.03 %)
6	http://perpustakaan.poltekkes-malang.ac.id/assets/file/kti/1602420024/12_LAMPIRAN.pdf	9 (0.03 %)
7	http://perpustakaan.poltekkes-malang.ac.id/assets/file/kti/1602420024/12_LAMPIRAN.pdf	7 (0.02 %)
8	http://referatu.net.ua/referats/7569/173568	7 (0.02 %)
9	http://repository.maranatha.edu/5780/2/0727029_AppendicesA.pdf	7 (0.02 %)
10	http://perpustakaan.poltekkes-malang.ac.id/assets/file/kti/1602420024/12_LAMPIRAN.pdf	7 (0.02 %)
Програма обміну базами даних (0.00 %)		☐
#	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
Інтернет (0.73 %)		☐
#	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://ontu.edu.ua/download/dissertation/review/2023/review-Verbivska.pdf	45 (9) (0.15 %)
2	http://perpustakaan.poltekkes-malang.ac.id/assets/file/kti/1602420024/12_LAMPIRAN.pdf	41 (6) (0.14 %)
3	https://ztu.edu.ua/ua/science/files/onelife/vysnovok_Naumchuk.pdf	30 (3) (0.10 %)
4	http://repository.ub.ac.id/125360/6/12_Lampiran.pdf	26 (4) (0.09 %)
5	http://referatu.net.ua/referats/7569/173568	18 (2) (0.06 %)
6	https://courses.cs.washington.edu/courses/cse410/22wi/lectures/03-binaryRepresentation.pdf	18 (3) (0.06 %)
7	https://iq.vntu.edu.ua/repository/getfile.php/9117.pdf	11 (1) (0.04 %)
8	http://repository.maranatha.edu/5780/2/0727029_AppendicesA.pdf	7 (1) (0.02 %)
9	https://svr.pnu.edu.ua/wp-content/uploads/sites/5/2018/04/dis_Rozmyrska.pdf	7 (1) (0.02 %)
10	https://lnu.edu.ua/wp-content/uploads/2025/01/novelty_Riy.pdf	6 (1) (0.02 %)
11	https://www.lute.lviv.ua/fileadmin/www.lac.lviv.ua/data/pidrozdliv/Aspirantura/Rady/Spec_vchena_rada/Dysertacii/2021_04/Ishchenko_1_opon.pdf	5 (1) (0.02 %)
☒ Список прийнятих фрагментів		
#	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)