Київський національний університет імені Тараса Шевченка
Факультет радіофізики, електроніки та комп'ютерних систем
Кафедра комп'ютерної інженерії

Дипломна робота на тему:

«Вирішення задачі розбиття множини за допомогою квантового комп'ютера»

Виконавець:
студент 4 року навчання
спеціальності 123 «Комп'ютерна інженерія»
Смичков Олексій Олегович

Науковий керівник:
к. ф.-м. н., доцент кафедри МТРФ
Іваненко Дмитро Олександрович

Рецензент:
д. ф-м. н., професор кафедри алгебри
механіко-математичного факультету
Шевченко Георгій Михайлович

До захисту допускаю _____ завідувач кафедрою **Бойко Юрій Володимирович**

Ухвалено на засіданні кафедри «____» _____ 2022 р., протокол № ___

Київ, 2022

Зміст

Вступ

Коли вчені та інженери стикаються зі складними розрахунковими проблемами, вони звертаються до суперкомп'ютерів. Однак існують певні класи задач, які не можуть вирішити навіть суперкомп'ютери.

Складні задачі — це задачі, які передбачають виконання великої кількості операцій для їх вирішення. Моделювання поведінки окремих атомів у молекулі є складною задачею, оскільки численна кількість електронів взаємодіє один з одним. Вибір ідеальних маршрутів для кількох сотень танкерів у глобальній судноплавній мережі також є складною задачею.

На відміну від класичних комп'ютерів, квантові комп'ютери використовують новий підхід — створюють багатовимірні простори, де виникають певні закономірності, що зв'язують окремі дані.

Наприклад, восьми біт достатньо, щоб класичний комп'ютер представляв будь-яке число від 0 до 255, однак восьми кубітів достатньо, щоб квантовий комп'ютер представляв кожне число від 0 до 255 одночасно.

До компаній, які розробляють обладнання для квантових комп'ютерів, відносяться Microsoft, IBM, Google, Intel, D-Wave Systems, Honeywell та ін. Багато хто з них працює спільно з дослідницькими групами провідних технічних інститутів, і разом вони досягають значних успіхів. Зараз у світі є близько десяти різних моделей квантових комп'ютерів, всі ці моделі є експериментальними та знаходяться на різних стадіях розробки, тому поки що жодна з них не дозволяє вирішувати задачі на практичному рівні.

Доступ до деяких квантових комп'ютерів можливий за допомогою хмарних сервісів. Майданчик IBM «Quantum Experience» дає доступ як до симуляторів квантових обчислень, що працюють на класичному комп'ютері, так і до самих квантових обчислювачів, на яких можна перевірити свої алгоритми. Для цього потрібно зареєструватися на сайті концерну.

Квантовий комп’ютер

Квантові обчислення — це технологія, яка використовує закони квантової механіки для зберігання даних і виконання обчислень.

Класичні комп’ютери кодують інформацію у двійкових «бітах», які можуть приймати значення 0 або 1, тобто їх операції засновані на одній з двох позицій стану, наприклад, «увімкнено» або «вимкнено», тобто комп’ютери цього типу виконують операції, використовуючи лише певне положення фізичного стану (біту).

У квантовому комп’ютері основною одиницею пам’яті є квантовий біт або кубіт. Квантові комп’ютери виконують обчислення на основі ймовірності стану об’єкта до того, як він буде виміряний, а не лише 1 або 0, що означає, що вони мають можливість обробляти одночасно потенційно більше даних у порівнянні з класичними комп’ютерами.

У квантових обчисленнях операції замість цього використовують квантовий стан об’єкта, що й є так званим кубітом. Ці стани є невизначеними властивостями об’єкта фізичної системи до того, як вони були виявлені, наприклад, спін електрона або поляризація фотона.

Замість того, щоб мати чітку позицію, невиміряні квантові стани можуть одночасно перебувати в багатьох різних системах, це явище змішаного стану називають «суперпозицією». Можна провести аналогію з монетою, що обертається в повітрі, перш ніж потрапити у вашу руку.

Ці суперпозиції можуть бути заплутані з суперпозиціями інших об’єктів, тобто їх кінцеві результати будуть математично пов’язані, навіть якщо ми ще не знаємо, що вони собою являють.

Кубіти також можуть бути нерозривно пов’язані між собою за допомогою явища, яке називається квантовою заплутаністю. Результатом є те, що серія кубітів може представляти різні речі одночасно.

Кубіти

Кубіт — найменша одиниця інформації в квантовому комп'ютері (аналог біта в звичайному комп'ютері), що використовується для квантових обчислень. Як і біт, кубіт допускає два власні стани, що позначаються $|0\rangle$ та $|1\rangle$ (позначення Дірака), але при цьому може бути і в їх суперпозиції. У загальному випадку його хвильова функція має вигляд $A|0\rangle + B|1\rangle$, де A та B називаються амплітудами ймовірностей і є комплексними числами, що задовольняють умові $|A|^2 + |B|^2 = 1$.

Стан кубіту зручно представляти як стрілку у сфері Блоха:

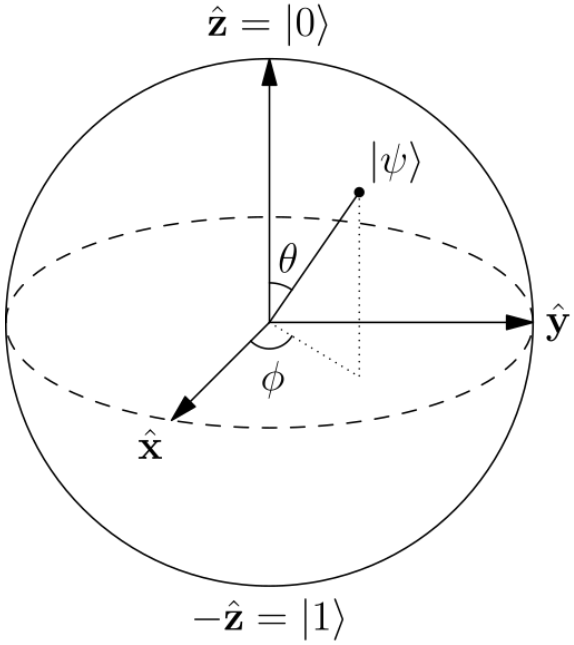


Рис. 1. Сфера Блоха.

Вимірюючи стан кубіту, можна отримати лише один з його власних станів. Ймовірність отримати кожен з них дорівнює $|A|^2$ і $|B|^2$ відповідно. Кубіти можуть бути заплутані один з одним. Квантову заплутаність можуть мати два і більше кубітів, яка виявляється у особливій кореляції між цими кубітами, що неможливим у класичних системах. Одним з найпростіших прикладів заплутаності двох кубітів є стан Белла $|\Phi^+\rangle$.

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

Запис $|00\rangle$ означає стан, коли обидва кубіти перебувають у стані $|0\rangle$. Для стану Белла характерно те, що під час вимірювань першого кубіту можливі два результати: 0 з ймовірністю $\frac{1}{2}$ і кінцевим станом $|\varphi'\rangle = |00\rangle$, і 1 з ймовірністю $\frac{1}{2}$ і кінцевим станом

$|\varphi''\rangle = |11\rangle$. Як наслідок, вимірювання другого кубіту завжди дає той же результат, що й вимір першого кубіту, тобто дані вимірів виявляються залежними або корельованими.

У той час, як для повного опису системи з n класичних бітів достатньо n нулів та одиниць, для опису системи з n кубітів необхідно $(2n-1)$ комплексних чисел. Це пов'язано з тим, що n-кубітну систему можна представити як вектор у 2^n-мірному Гільбертовому просторі. Звідси випливає, що система з кубітів може вмістити експоненційно більше інформації, ніж система з бітів.

Наприклад, в один кубіт можна записати до двох бітів інформації Шеннона, використовуючи надщільне кодування, а система з n кубітів може використовуватися для кодування 2^n чисел, що застосовується, наприклад, в квантовому машинному навчанні.

Однак, варто враховувати, що експоненційне збільшення простору станів системи не обов'язково призводить до експоненційного зростання обчислювальної потужності у зв'язку зі складністю кодування та зчитування інформації.

Q# та Azure Quantum

QSharp — це open source мова програмування, розроблена корпорацією Microsoft для реалізації квантових алгоритмів. Він є складовою пакету «Quantum Development Kit», який також має додаткові бібліотеки Q#, симулятори, доповнення до інших мов програмування та документацію до підтримуваних API. Окрім стандартної, пакет містить бібліотеки для числових розрахунків, машинного навчання та квантової хімії.

Програму Q# можливо скомпілювати в незалежну програму, або викликати з іншої (головної) програми, написаної іншою мовою (Python чи C#). Під час компіляції програми та її запуску середовище створює екземпляр квантового симулятора, який отримає відповідний код Q#. Симулятор використовує цей код для ініціалізації кубітів і виконує над ними вказані операції. Результат роботи квантового симулятору повертаються до програми після закінчення виконання.

Azure Quantum — це хмарна служба Майкрософт, яка була створена для надання віддаленим користувачам доступу до квантових обчислень. Завдяки Azure Quantum користувачі мають доступ до програмного забезпечення, яке дозволяє клієнтам писати код, який може працювати на квантовому обладнанні.

Квантові обчислення та оптимізація — це два незалежні елементи служби «Azure Quantum». Обидва можна налаштувати в Azure, створивши ресурс «Quantum Workspace». В результаті Azure Quantum суттєво скорочує час розробки, дозволяючи розробникам створити квантове рішення один раз і запустити його на різних комп’ютерах з мінімальними змінами.

Квантова оптимізація

У задачі оптимізації ми шукаємо найкращу з багатьох можливих комбінацій. Проблеми з оптимізацією включають проблеми планування, наприклад «Чи слід відправити цей пакет на цій вантажівці чи на наступній?» або «Який найефективніший маршрут повинен пройти комівояжер, щоб відвідати різні міста?»

Фізика може допомогти вирішити подібні проблеми, оскільки ми можемо сформулювати їх як проблеми мінімізації енергії. Основне правило фізики полягає в тому, що все прагне до мінімального енергетичного стану. Така поведінка справедлива і в світі квантової фізики. Квантовий відпал просто використовує квантову фізику, щоб знайти низькоенергетичні стани проблеми і, отже, оптимальну або майже оптимальну комбінацію елементів.

Вибірка з багатьох низькоенергетичних станів і характеристика форми енергетичного ландшафту корисні для задач машинного навчання, де потрібно побудувати імовірнісну модель реальності. Зразки надають інформацію про стан моделі для заданого набору параметрів, які потім можна використовувати для покращення моделі.

Як працює квантовий відпал

Як і у випадку з класичними бітами, кубіт може приймати стан 0 або 1. Але, оскільки кубіт є квантовим об’єктом, він також може перебувати в суперпозиції стану 0 і 1 одночасно. В кінці відпалу під час заміру стану кубіти коллапсують до класичний станів.

Фізику цього процесу можна візуалізувати за допомогою енергетичної діаграми, як на рисунку 7. Ця діаграма змінюється з часом, як показано в (а), (б) і (в). Спочатку існує лише одна «западина» (а) з єдиним мінімумом. Потім запускається процес квантового відпалу, бар’єр піднімається і перетворює енергетичну діаграму в «подвійну ямку» (б). Тут нижня точка лівої «западини» відповідає стану 0, а нижня точка правої — стану 1. Наприкінці відпалу кубіт потрапляє в одну з цих «западин» (в).

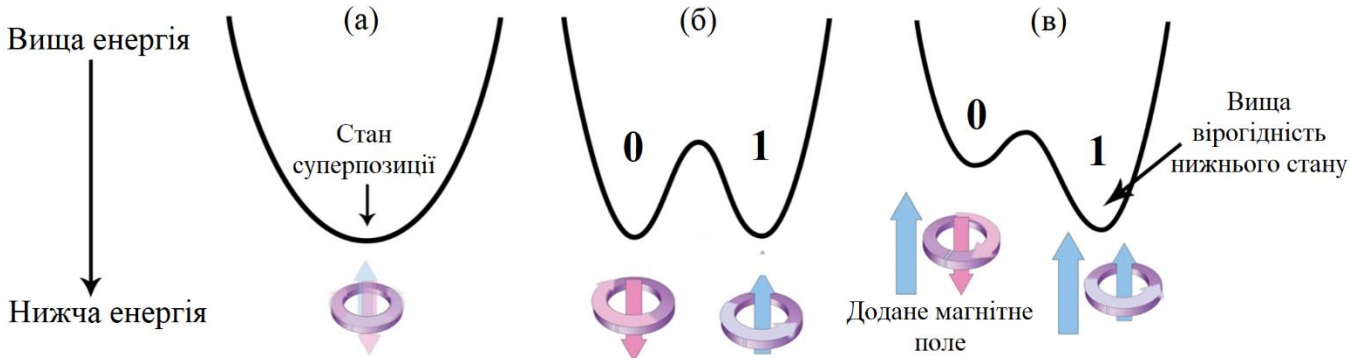


Рис. 2. Енергетична діаграма змінюється з часом у міру запуску процесу квантового відпалу та застосування зміщення.

За рівних умов ймовірність того, що кубіт коллапсує в 0 або 1, дорівнює 50%. Однак можливо контролювати ймовірність його потрапляння в стан 0 або 1, застосувавши зовнішнє магнітне поле до кубіта. Це поле змінює потенціал подвійної ямки, збільшуючи ймовірність того, що кубіт опиниться в западині мінімуму. Програмована величина, яка керує зовнішнім магнітним полем, називається зміщенням.

Використання з’єднувача дозволяє реалізувати явище квантової заплутаністю. Коли два кубіти заплутані, їх можна розглядати як один об’єкт з чотирма можливими станами. Рисунок 8 ілюструє цю ідею, показуючи потенціал з чотирма станами, кожен з яких відповідає різній комбінації двох кубітів: (0, 0), (0, 1), (1, 1) і (1, 0). Відносна енергія кожного стану залежить від зміщень кубітів і зв’язку між ними. Під час відпалу стани

кубітів потенційно розміщуються в ландшафт, перш ніж остаточно встановитися в (1, 1) наприкінці відпалу.

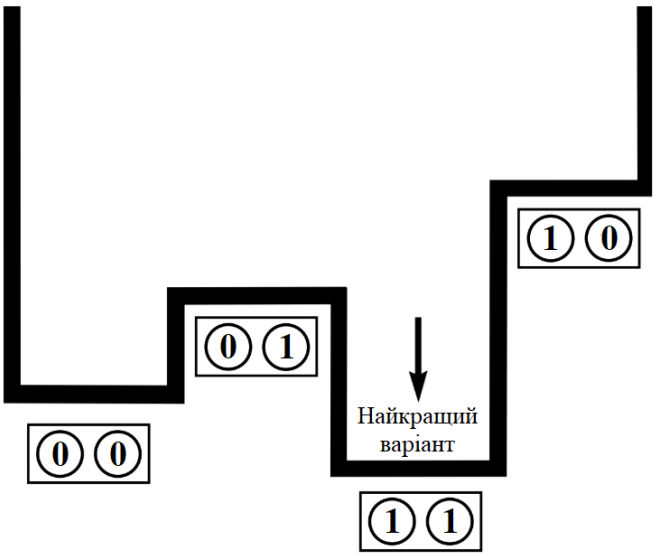


Рис. 3. Енергетична діаграма, що показує найкраще рішення.

При формулюванні проблеми користувач вибирає значення для зміщень і зв'язків. Зміщення та зв'язки визначають енергетичний ландшафт, і квантовий комп'ютер знаходить мінімальну енергію цього ландшафту. Це і є сутність квантового відпалу.

Системи стають дедалі складнішими з додаванням кубітів: два кубіти мають чотири можливі стани, над якими можна визначити енергетичний ландшафт; три кубіти мають вісім. Кожен додатковий кубіт подвоює кількість станів, за якими можна визначити енергетичний ландшафт: кількість станів зростає експоненційно разом із кількістю кубітів.

Підсумовуючи, системи починаються з набору кубітів, кожен з яких знаходиться у стані суперпозиції 0 і 1. Коли починається квантовий відпал, вводяться зв'язки та зміщення, кубіти заплутуються. На цьому етапі система перебуває в заплутаному стані багатьох можливих варіантів. В кінці відпалу кожен кубіти займуть класичні стани, які представлятимуть мінімальне рішення проблеми, або близьке до нього.

Гамільтоніан

Класичний гамільтоніан — це математичний опис деякої фізичної системи в термінах енергій. Як приклад класичної системи розглянемо надзвичайно просту систему столу і яблука. Ця система має два можливі стани: яблуко на столі та яблуко на підлозі. В даному випадку гамільтоніан представляє енергетичний стан системи, з якого ми можемо зрозуміти, що стан яблука на столі має вищу енергію, ніж стан, коли яблуко лежить на підлозі.

Для квантової системи гамільтоніан є функцією, яка відображає певні (власні) стани в енергії. Тільки коли система знаходиться у власному стані, її енергія добре визначена гамільтоніаном. Коли система знаходиться в будь-якому іншому стані, її енергія невизначена. Набір власних станів із визначеними власними енергіями складає власний спектр.

Для квантового комп'ютера гамільтоніан можна представити у наступному вигляді:

$$\mathcal{H}_{ising} = -\frac{A(s)}{2}\left(\sum_i \hat{\sigma}_x{}^{(i)}\right) + \frac{B(s)}{2}\left(\sum_i h_z \hat{\sigma}_z{}^{(i)} + \sum_{i>j} J_{i,j}\hat{\sigma}_z{}^{(i)}\hat{\sigma}_z{}^{(i)}\right)$$

де $\hat{\sigma}_{x,z}{}^{(i)}$— це матриці Паулі, що виконуються над кубітом q_i, а h_i та j_i — зміщення кубіта та сила зв'язку. Ненульові значення h_i і $J_{i,j}$ обмежені тими, які доступні в робочому графіку. (1)

Гамільтоніан є сумою двох доданків: початкового і кінцевого гамільтоніана. Початковий гамільтоніан (перший член) — найнижчий енергетичний початковий стан, коли всі кубіти знаходяться в стані суперпозиції 0 і 1. Цей термін також називають тунельним гамільтоніаном. Кінцевий гамільтоніан (другий член) — найнижчий кінцевий енергетичний стан, він і є відповіддю на задачу, яку ми намагамося вирішити. Кінцевий стан є класичним, він включає зміщення кубітів і зв'язки між кубітами.

Математична реалізація

Розглянемо реалізацію алгоритму квантового відпалу на прикладі задачі розбиття множини чисел. Суть цієї задачі полягає в пошуку такого варіанту розбиття множини n натуральних чисел S на m підмножин $S_1, S_2, \dots, S_m$, щоб суми чисел відповідних підмножин були рівними (або якомога рівнішими).

Позначимо числа як $w_1, w_2, \dots, w_n$, суми чисел множин — як $W_0, W_1, W_2, \dots, W_m$ для $S, S_1, S_2, \dots, S_m$ відповідно. Виразимо суму елементів множини:

Для вирішення поставленої задачі ми будемо використовувати PUBO формулювання функції вартості, де змінні прийматимуть значення 0 або 1. Щоб знайти найкращий розподіл, необхідно сформулювати функцію вартості, яка зможе «відсіяти» всі непідхожі конфігурації за допомогою призначення їм вищої вартості. Таким чином, наша функція вартості матиме наступні складові:

- Штраф (збільшення вартості конфігурації) за відхилення від теоретичного рівного розподілу, де рівний розподіл — це сума чисел, поділена на кількість множин.
- Штраф за призначення одного і того ж числа кільком множинам, або не призначення жодній.

Позначимо теоретичного рівний розподіл як *D* та виразимо його значення:

$$D = \frac{(w_1 + w_2 + \cdots + w_n)}{m} = \frac{\sum_{\{w_j \in W_0\}} w_j}{m} = \frac{W_0}{m}$$

Штраф за відхилення від теоретичного рівного розподілу для множини виглядатиме наступним чином (значення функції вартості має бути додатнім, тому піднесемо в квадрат):

$$H_i = \left(\sum_{\{w_j \in W_i\}} w_j - D \right)^2 = (W_i - D)^2$$

Перша складова функції вартості матиме наступний вигляд:

$$H_A = H_1 + H_2 + \cdots + H_m = \sum_{i=1}^{m} H_i = \sum_{i=1}^{m} (W_i - D)^2$$

Введемо двійкову змінну x_i, яка буде відповідати за наявність даного числа в певній множині ($x_i = 1$), або його відсутність ($x_i = 0$). Для цього необхідно призначити єдину послідовність x_i для всіх множин підряд з наскрізною індексацією, відповідно до повторюваного списку чисел *S* для кожної підмножини:

	S_1				S_2				S_3			
Числа	w_1	w_2	...	w_n	w_{n+1}	w_{n+2}	...	w_{2n}	$w_{(m-1)n+1}$	$w_{(m-1)n+2}$	...	w_{mn}
Змінні призначення	x_1	x_2	...	x_5	x_{n+1}	x_{n+2}	...	x_{2n}	$x_{(m-1)n+1}$	$x_{(m-1)n+2}$	...	x_{mn}

Список чисел повторюється для кожної підмножини, тому:

$$w_1 = w_{n+1} = \cdots = w_{(m-1)n+1}$$

$$w_2 = w_{n+2} = \cdots = w_{(m-1)n+2}$$

...

$$w_n = w_{2n} = \cdots = w_{mn}$$

Визначимо доданки H_A:

$$H_1 = (w_1 x_1 + w_2 x_2 + \cdots + w_n x_n - D)^2$$

$$H_2 = (w_1 x_{n+1} + w_2 x_{n+2} + \cdots + w_n x_{2n} - D)^2$$

...

$$H_m = \left(w_1 x_{(m-1)n+1} + w_2 x_{(m-1)n+2} + \cdots + w_n x_{mn} - D\right)^2$$

Згорнемо:

$$H_i = \left(\sum_{j=(i-1)n+1}^{in} w_j x_j - D \right)^2$$

Маємо загалом:

$$H_A = \sum_{i=1}^{m} \left(\sum_{j=(i-1)n+1}^{in} w_j x_j - D \right)^2$$

Тепер необхідно ввести обмеження на конфігурації, в яких одне число потрапляє в декілька множин, або не попадає в жодну. Використовуючи вище наведені позначення, можна скласти функції вартості для чисел:

$$H_1' = \left(w_1 x_1 + w_{n+1} x_{n+1} + \cdots + w_{(m-1)n+1} x_{(m-1)n+1} - w_1 \right)^2$$

$$H_2' = \left(w_2 x_2 + w_{n+2} x_{n+2} + \cdots + w_{(m-1)n+2} x_{(m-1)n+2} - w_2 \right)^2$$

...

$$H_n' = (w_n x_n + w_{2n} x_{2n} + \cdots + w_{mn} x_{mn} - w_n)^2$$

Маємо формулу:

$$H_i' = \left(\sum_{j=0}^{m-1} w_{i+jn} x_{i+jn} - w_n \right)^2$$

Друга складова функції вартості матиме наступний вигляд:

$$H_B = H_1' + H_2' + \cdots + H_n' = \sum_{i=1}^{n} H_i' = \sum_{i=1}^{n} \left(\sum_{j=0}^{m-1} w_{i+jn} x_{i+jn} - w_n \right)^2$$

Отримаємо загальну функцію вартості для нашої задачі:

$$H = H_A + H_B = \sum_{i=1}^{m} \left(\sum_{j=(i-1)n+1}^{in} w_j x_j - D \right)^2 + \sum_{i=1}^{n} \left(\sum_{j=0}^{m-1} w_{i+jn} x_{i+jn} - w_n \right)^2$$

Для наочності розглянемо приклад $n = 5, m = 3$. Отримаємо наступний вигляд функції вартості для першої множини:

$$H_1 = (w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 - D)^2$$

Для спрощення введемо заміну:

$$w_1x_1 = a;\ w_2x_2 = b; w_3x_3 = c; w_4x_4 = d; w_5x_5 = e; D = f$$

Відкриємо дужки:

$$H_1 = (a + b + c + d + e - f) = a^2 + b^2 + c^2 + d^2 + e^2 + f^2 +$$

$$+2(ab + ac + ad + ae + bc + bd + be + cd + ce + de) - 2(af + bf + cf + df + ef)$$

Також складемо функцію вартості для першого числа:

$$H_1' = (w_1x_1 + w_6x_6 + w_{11}x_{11} - w_1)^2$$

Насправді w_1, w_6, w_{11} позначають одне і те саме число, представлене для різних множин, тому:

$$H_1' = (w_1x_1 + w_1x_6 + w_1x_{11} - w_1)^2$$

Розкриємо дужки, згрупуємо та отримаємо наступні терми:

$$H_1' = {w_1}^2{x_1}^2 + {w_1}^2{x_6}^2 + {w_1}^2{x_{11}}^2 + {w_1}^2 + 2({w_1}^2x_1x_6 + {w_1}^2x_1x_{11} + {w_1}^2x_6x_{11}) -$$

$$-2({w_1}^2x_1 + {w_1}^2x_5 + {w_1}^2x_{11})$$

Лістинг коду

Перейдемо до написання програми мовою «Python». Спочатку ми повинні створити екземпляр об’єкта Workspace, який дозволить нам підключитися до AzureQuantum:

```
workspace = Workspace (
    subscription_id = "758db824-e601-4c8e-9e42-dabd84126346",
    resource_group = "azurequantum",
    name = "quantumdiploma",
    location = "eastus"
)
```

Тепер необхідно написати функцію, яка додаватиме терми відповідно до нашого визначення H_A, де ми розраховували вартість для розподілу:

$$H_1 = a^2 + b^2 + c^2 + d^2 + e^2 + f^2 +$$

$$+2(ab + ac + ad + ae + bc + bd + be + cd + ce + de) - 2(af + bf + cf + df + ef)$$

```
def addEqualDistributionTerms(cycledNumbers, mainSetLen, setsCount, eqDistrib):
    terms = []
    for i in range(setsCount):
        for ind, num in enumerate(np.array_split(cycledNumbers, setsCount)[i], i * mainSetLen):
            terms.append(Term(c = pow(num, 2), indices=[ind]))
            terms.append(Term(c = -2 * num * eqDistrib, indices = [ind]))
        for comb in combinations(range(i * mainSetLen, (i + 1) * mainSetLen), 2):
            num1 = cycledNumbers[comb[0]]
            num2 = cycledNumbers[comb[1]]
            terms.append(Term(c = 2 * num1 * num2, indices = [comb[0], comb[1]]))
    return terms
```

Далі напишемо функцію для формування термів відповідно до H_B, чим обмежимо непідхожі конфігурації:

$$H_1' = {w_1}^2{x_1}^2 + {w_1}^2{x_6}^2 + {w_1}^2{x_{11}}^2 + {w_1}^2 + 2({w_1}^2x_1x_6 + {w_1}^2x_1x_{11} + {w_1}^2x_6x_{11}) -$$

$$-2({w_1}^2x_1 + {w_1}^2x_5 + {w_1}^2x_{11})$$

```
def addPenalizingTerms(cycledNumbersGrouped):
    terms = []
    for i in range(len(mainSet)):
        initialInd = i * setsCount
        finalInd = (i + 1) * setsCount
        for j in range(initialInd, finalInd):
            num = cycledNumbersGrouped[j][0]
            ind = cycledNumbersGrouped[j][1]
            terms.append(Term(c = pow(num, 2), indices = [ind]))
        for comb in combinations(range(initialInd, finalInd), 2):
            num = cycledNumbersGrouped[comb[0]][0]
            firstNumInd = cycledNumbersGrouped[comb[0]][1]
            secondNumInd = cycledNumbersGrouped[comb[1]][1]
            terms.append(Term(c = 2 * pow(num, 2), indices = [firstNumInd, secondNumInd]))
        for j in range(initialInd, finalInd):
            num = cycledNumbersGrouped[j][0]
            ind = cycledNumbersGrouped[j][1]
            terms.append(Term(c = -2 * pow(num, 2), indices = [ind]))
    terms.append(Term(c = cycledNumbersGrouped[initialInd][0] * cycledNumbersGrouped[initialInd][0], indices=[]))
    return terms
```

В загальному отримаємо функції вартості для нашої «проблеми»:

```
def createSetDistributionProblem(mainSet, setsCount, problemName):
    terms = []
    cycledNumbers = []
    cycledNumbersGrouped = []
    setSum = 0
    finalInd = -1
    eqDistrib = 0
    for i in range (len(mainSet)):
        setSum += mainSet[i]
    eqDistrib = setSum / setsCount
    print("Початкова множина: {", end = "")
    for i in range(len(mainSet)):
        if i != len(mainSet) - 1:
            print(str(mainSet[i]), end = ", ")
        else:
            print(str(mainSet[i]), end = "}\n")
    print(f"\n{problemName}.")
    print("Рівне (теоретичне) розбиття: " + str(eqDistrib), end = "\n\n")
    cycledNumbers = mainSet * setsCount
    for i in range(len(mainSet)):
        for j in range(setsCount):
            k = i + j * len(mainSet)
            cycledNumbersGrouped.append([cycledNumbers[i], k])
    terms = addEqualDistributionTerms(cycledNumbers, len(mainSet), setsCount, eqDistrib) +
        addPenalizingTerms(cycledNumbersGrouped)
    return terms
```

Також необхідно створимо функцію, яка інтерпретує результати, які нам поверне «solver»:

```
def printResults(problemResults, cycledNumbers, setsCount):
    print("\rРезультат роботи: \n")
    print("Запис: \"номер_признач_числа(число)\"\n")
    resultsList = list(problemResults['configuration'].values())
    for i in range(setsCount):
        partResultsList = np.array_split(resultsList, setsCount)[i]
        partNumbers = np.array_split(cycledNumbers, setsCount)[i]
        resultStr = ""
        setSum = 0
        for j in range(len(partResultsList)):
            if partResultsList[j] == 1:
                resultStr += f"{str(j + 1)}({str(partNumbers[j])}), "
                setSum += partNumbers[j]
        if len(resultStr) > 2:
            resultStr = resultStr[0:-2]
        print(f"Множина {str(i + 1)}: {resultStr}; сума чисел множини: {str(setSum)}.")
    print("")
```

Ініціалізація та виконання:

```
mainSet = []
for i in range(20):
    mainSet.append(rd.randint(1, 100))
setsCount = 6
problemName = f"Розбиття множини з {str(len(mainSet))} чисел на {str(setsCount)} підмножин"
terms = createSetDistributionProblem(mainSet, setsCount, problemName)
problem = Problem(name = problemName, problem_type = ProblemType.pubo, terms = terms)
startTime = time.time()
solver = SimulatedAnnealing(workspace, timeout = 300)
results = solver.optimize(problem)
endTime = time.time()
printResults(results, mainSet * setsCount, setsCount)
print("Витрачено часу: " + str(endTime - startTime))
```

Тестування отриманої програми

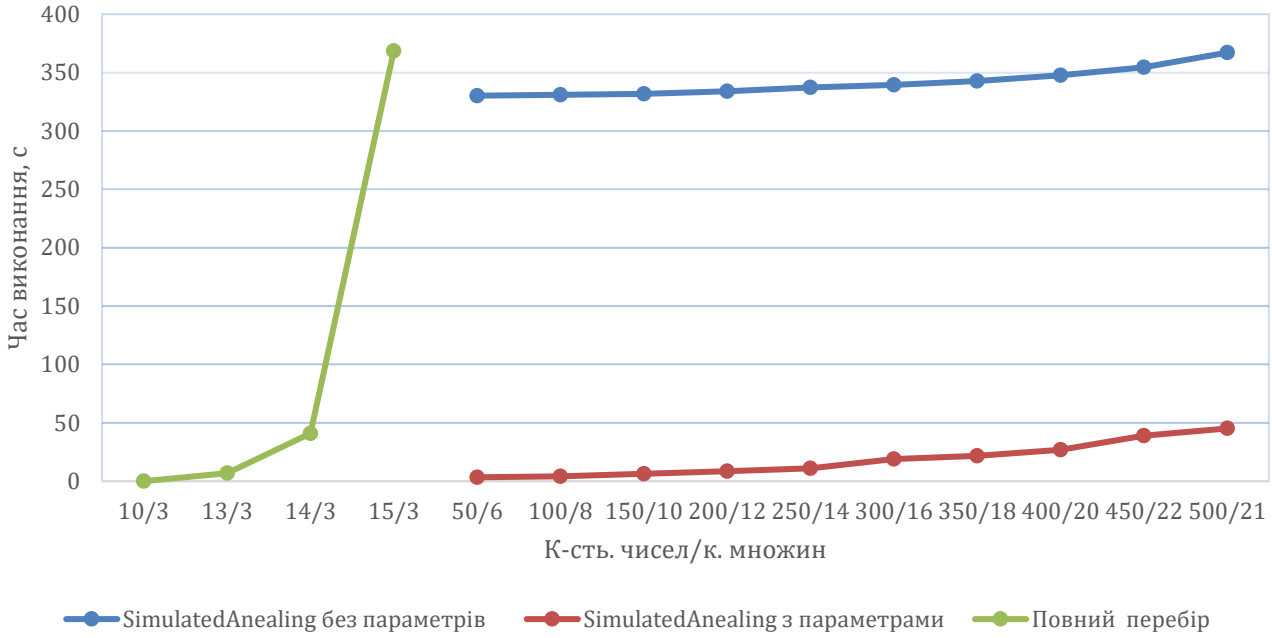
Перевіримо правильність роботи програми, використаємо невелику кількість чисел та множин для наочності:

```
Початкова множина: {99, 77, 52, 92, 81, 31, 95, 35, 60, 9, 84, 100, 98, 63, 68, 41, 26, 18, 11, 55}

Розбиття множини з 20 чисел на 6 підмножин.
Рівне (теоретичне) розбиття: 199.16666666666666

Результат роботи:

Запис: "номер_признач_числа(число)"

Множина 1: 1(99), 12(100); сума чисел множини: 199.
Множина 2: 3(52), 6(31), 13(98), 18(18); сума чисел множини: 199.
Множина 3: 4(92), 8(35), 10(9), 14(63); сума чисел множини: 199.
Множина 4: 2(77), 5(81), 16(41); сума чисел множини: 199.
Множина 5: 7(95), 15(68), 17(26), 19(11); сума чисел множини: 200.
Множина 6: 9(60), 11(84), 20(55); сума чисел множини: 199.
```

```
Початкова множина: {52, 83, 59, 7, 4, 28, 37, 34, 70, 88, 81, 44, 78, 7, 90, 34, 28, 66, 85, 52, 53, 23, 55, 43,
77, 81, 31, 86, 73, 15, 49, 19}

Розбиття множини з 32 чисел на 8 підмножин.
Рівне (теоретичне) розбиття: 204.0

Результат роботи:

Запис: "номер_признач_числа(число)"

Множина 1: 2(83), 12(44), 13(78); сума чисел множини: 205.
Множина 2: 1(52), 3(59), 5(4), 15(90); сума чисел множини: 205.
Множина 3: 7(37), 27(31), 28(86), 31(49); сума чисел множини: 203.
Множина 4: 8(34), 14(7), 17(28), 20(52), 26(81); сума чисел множини: 202.
Множина 5: 4(7), 6(28), 9(70), 22(23), 25(77); сума чисел множини: 205.
Множина 6: 10(88), 24(43), 29(73); сума чисел множини: 204.
Множина 7: 11(81), 16(34), 23(55), 30(15), 32(19); сума чисел множини: 204.
Множина 8: 18(66), 19(85), 21(53); сума чисел множини: 204.
```

```
Початкова множина: {24, 4, 42, 89, 17, 98, 99, 85, 40, 83, 48, 55, 58, 75, 17, 67, 17, 34, 33, 83, 34, 78, 8, 61,
1, 100, 47, 55, 11, 88, 40, 80, 71, 37, 46, 84, 64, 18, 34, 100}

Розбиття множини з 40 чисел на 5 підмножин.
Рівне (теоретичне) розбиття: 425.0

Результат роботи:

Запис: "номер_признач_числа(число)"

Множина 1: 2(4), 4(89), 9(40), 15(17), 16(67), 22(78), 35(46), 36(84); сума чисел множини: 425.
Множина 2: 3(42), 5(17), 10(83), 23(8), 24(61), 25(1), 27(47), 28(55), 29(11), 40(100); сума чисел множини: 425.
Множина 3: 6(98), 18(34), 31(40), 32(80), 33(71), 34(37), 37(64); сума чисел множини: 424.
Множина 4: 7(99), 11(48), 12(55), 14(75), 19(33), 20(83), 39(34); сума чисел множини: 427.
Множина 5: 1(24), 8(85), 13(58), 17(17), 21(34), 26(100), 30(88), 38(18); сума чисел множини: 424.
```

Оцінимо швидкість роботи програми; порівняємо час роботи «солвера» з різними параметрами та повним перебором:



Найкращі показники швидкості виконання (вказані на графіку) отримано з наступними параметрами:

```
SimulatedAnnealing(workspace, sweeps=5, beta_start=1, beta_stop=0, restarts=3, timeout = 100)
```

Висновок

В ході виконання дипломної роботи було опановано принцип функціонування квантового комп’ютера, основні засади квантової оптимізації та відповідні математичні моделі. Також було описано принципи формулювання задачі для квантового відпалу за допомогою функції витрат (гамільтоніану).

Практична части дипломної роботи включає в себе реалізацію задачі розбиття множини за допомогою PUBO формулювання функції витрат. Код програми написаний мовою Python, що може бути зручним для передачі результату виконання зовнішнім програмам для подальшого використання.

Також варто зазначити, що не будь-яку оптимізаційну задачу варто вирішувати за допомогою квантового комп’ютера. Іноді доцільно використовувати класичні алгоритми, оскільки їх реалізація, як правило, простіша за квантову, а перевага в швидкості не завжди є суттєвою. Натомість квантову оптимізації слід застосовувати у випадку надважких задач з широким простором пошуку та великою кількістю локальних мінімумів. Окрім цього, ефективність застосування квантової оптимізації сильно залежить від правильного підбору параметрів вирішувача.

Список використаних джерел

Quantum computing for computer scientists; N. Yanofsky and M. Mannucci — Cambridge Press, 2008 p.

What are the Q# programming language and the Quantum Development Kit? [веб-сайт] — URL: https://docs.microsoft.com/en-us/azure/quantum/overview-what-is-qsharp-and-qdk

What is Azure Quantum? [веб-сайт] — URL: https://docs.microsoft.com/en-us/azure/quantum/overview-azure-quantum

Adiabatic Quantum Computation Is Equivalent to Standard Quantum Computation; Dorit Aharonov, Wim van Dam, Julia Kempe — Article in SIAM Review, 2008

What is Quantum Annealing? [веб-сайт] — URL: https://docs.dwavesys.com/docs/latest/c_gs_2.html

Microsoft QIO provider [веб-сайт] — URL: https://docs.microsoft.com/en-us/azure/quantum/provider-microsoft-qio

Submit optimization jobs to Azure Quantum [веб-сайт] — URL: https://docs.microsoft.com/en-us/azure/quantum/how-to-submit-jobs-optimization?pivots=ide-portal