УДК 004.056.55

ПОВЫШЕНИЕ БЫСТРОДЕЙСТВИЯ КВАНТОВОГО АЛГОРИТМА ГРОВЕРА ПУТЕМ ПРИМЕНЕНИЯ ИНВЕРСИИ ВОКРУГ СРЕДНЕГО

Маслов И. О., Бачило А. О., Черкесова Л. В.

Донской государственный технический университет, Ростов-на-Дону, Российская Федерация
maslik220@gmail.com
a-bachilo@mail.ru
chia2002@inbox.ru

Произведена реализация алгоритма Гровера на языке программирования Python с помощью облачного квантового компьютера сервиса Riggeti Forest. Рассмотрена инверсия вокруг среднего, которая позволяет решить задачу поиска за время порядка квадратного корня. Это обеспечивает большой потенциал для практического применения квантового алгоритма Гровера. Полученный модифицированный алгоритм Гровера отличается более высокой производительностью и быстродействием при осуществлении поиска.

Ключевые слова: квантовый алгоритм Гровера, кубит, оракул, инверсия вокруг среднего, суперпозиция, Python.

UDC 004.056.55

SPEED IMPROVEMENT OF QUANTUM GROVER'S ALGORITHM THROUGH THE USE OF INVERSION ABOUT THE MEAN

Maslov I.O., Bachilo A.O., Cherkesova L.V.

Don State Technical University, Rostov-on-Don, Russian Federation
maslik220@gmail.com
chia2002@inbox.ru
a-bachilo@mail.ru

The article provides the implementation of Grover's algorithm in the Python programming language, using the Riggeti Forest cloud quantum computer. The authors of this article considered the inversion about the mean, which allows them to solve the search problem quadratically faster than for any classical counterpart. In turn, this provides great potential for the practical application of Grover's quantum algorithm. The resulting modified Grover's algorithm is characterized by higher performance and speed in the search.

Key words: Grover's quantum algorithm, qubit, oracle, inversion about the mean, superposition, Python.

Введение. Квантовый алгоритм представляет собой классический алгоритм, который задает последовательность унитарных операций (гейтов или вентилей) с указанием, над какими именно кубитами их надо совершать. Квантовый алгоритм задается либо в виде словесного описания таких команд, либо с помощью их графической записи в виде системы вентилей (quantum gate array).

Результат работы квантового алгоритма носит вероятностный характер. За счёт небольшого увеличения количества операций в алгоритме можно сколь угодно приблизить вероятность получения правильного результата к единице. Любая задача, решаемая квантовым алгоритмом, может быть решена и классическим компьютером путём прямого вычисления унитарных матриц экспоненциальной размерности, получения явного вида квантовых состояний. В частности, проблемы, неразрешимые на классических компьютерах

(например, проблема остановки), остаются неразрешимыми и на квантовых. Но такое прямое моделирование требует экспоненциального времени, и потому возникает возможность, используя квантовый параллелизм, ускорять на квантовом компьютере некоторые классические алгоритмы.

Алгоритм Гровера — алгоритм квантового поиска, время выполнения которого во много раз быстрее, чем у классических. Алгоритм Гровера не предназначен для нахождения элемента в базе данных, его целью является поиск по входам функции, чтобы проверить, возвращает ли функция значение «true» для определенных входных данных. Это полезный метод в случае, если функция неизвестна или чрезвычайно сложна, и мы хотим найти, для каких входных значений функция возвращает истину или дает правильный вектор решения уравнения.

В этой статье узнаем, что это такое и как реализовать алгоритм поиска Гровера. Сначала опишем алгоритм Гровера и представим квантовую схему. Затем продолжим реализацию алгоритма на платформах Python и Rigetti Forest.

Цель исследования. Исходя из вышесказанного, необходимо доказать преимущество квантового алгоритма перед алгоритмами других вычислительных моделей. Алгоритм Гровера состоит из двух главных частей — это построение ворот Адамара и инверсии вокруг среднего. Квантовые алгоритмы до сих пор остаются очень сложны к пониманию, поэтому в данной статье мы предоставим код программы на языке программирования Python, чтобы каждый смог наглядно увидеть, что происходит в данном алгоритме.

Алгоритм поиска Гровера. По своей сути это задача неструктурированного поиска. Если есть неупорядоченный набор данных («стог сена») и требуется найти в нём какой-то один элемент, удовлетворяющий специфическому требованию (этот элемент один — «иголка»), алгоритм Гровера поможет, поскольку альтернативой является классический перебор вариантов. Например, если рассмотреть аналогию с базой данных автомобилистов с именами, упорядоченными по алфавиту (пусть в ней будет взаимно однозначное соответствие между фамилией и номером автомобиля), то упорядоченным поиском является поиск номера автомобиля по фамилии. А неупорядоченным поиском является обратная задача — поиск фамилии по номеру автомобиля. В данной аналогии оракулом является функция (преобразованная соответствующим образом), которая по фамилии возвращает номер автомобиля. Тогда эта задача сводится к задаче Гровера при помощи кодирования фамилий в бинарное представление, а сама функция возвращает ответ на вопрос «владеет ли данный автомобилист N автомобилем X?», где N — входной параметр функции, а X — параметр алгоритма, как бы «зашиваемый» внутрь оракула при его построении. Соответственно, этот оракул возвращает значение 1 («да») только единственно для той фамилии, напротив которой в базе данных стоит искомый номер автомобиля.

Представьте, что нам дана функция $f : \{0,1\}^n \to \{0,1\}$ в виде набора логических операторов «И» и «ИЛИ». Функция возвращает «true» только для одной двоичной строки из нулей и единиц. Кодирование такой функции может быть относительно простым. Однако для выяснения, для какой комбинации нулей и единиц функция возвращает «true», в худшем случае потребуется 2^n вызовов функций на классической машине. На квантовом компьютере мы можем выразить f как действительный набор квантовых логических элементов, составляющих оракул (непреложная истина) U_f, и использовать алгоритм поиска Гровера,

чтобы найти правильный вход с очень высокой точностью только в $\sqrt{2^n}$ итерациях. Это квадратичное ускорение.

Например, представим некую функцию. В классическом стиле функция выражена:

$$f(x) = \begin{cases} 1, x = 10 \\ 0, x \neq 10 \end{cases} \quad (1)$$

Затем нужно представить функцию $f : \{0,1\}^n \to \{0,1\}$ в виде квантового оракула U_f. Это немного сложнее, так как первое предположение $U_f|x\rangle = |f(x)\rangle$ не является унитарной и обратимой операцией, поэтому это не квантовый вход ($|x\rangle$ состоит из двух кубитов, тогда как $|f(x)\rangle$ только из одного кубита). Однако существует метод построения квантового оракула путем добавления контрольного кубита и его подготовки в состоянии $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, так что действие U_f:

$$U_f|x\rangle\,|-\rangle = (-1)^{f(x)}|x\rangle|-\rangle \quad (2)$$

Поворот знака амплитуды в случае f(x) = 1 необходим. В нашем анализе отбросим контрольный кубит и выразим действие квантового оракула как:

$$U_f|x\rangle = (-1)^{f(x)}|x\rangle \quad (3)$$

Квантовая схема представлена ниже:

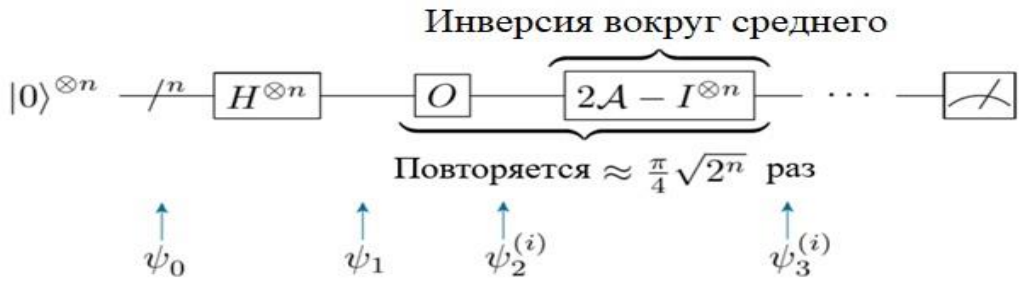

Рис. 1. Схема инверсии вокруг среднего

Одним словом, состояние системы устанавливается в виде суперпозиции всех возможных входных данных, затем вероятность нахождения искомого входного сигнала увеличивается на каждой итерации алгоритма. В следующем разделе проанализируем каждый шаг алгоритма на основе примера на Python с использованием платформы Rigetti Forest.

Для начала импортируем все необходимые объекты:

```
import numpy as np
from pyquil.quil import Program
from pyquil.api import QVMConnection
from pyquil.gates import H, I
```

Квантовый оракул. Создадим квантовый оракул, который помечает искомую строку. Продолжаем предыдущий пример с найденной строкой длины 10. Обратите внимание, что это только для целей представления, весь смысл в том, что квантовый оракул уже задан алгоритму, и мы пытаемся определить, для какого входа функция возвращает «true».

Это простой случай n = 2.

```
SEARCHED_STRING = "10"
N = len(SEARCHED_STRING)
```

```
oracle = np.zeros(shape=(2 ** N, 2 ** N))
for b in range(2 ** N):
if np.binary_repr(b, N) == SEARCHED_STRING:
oracle[b, b] = -1
else:
oracle[b, b] = 1
print(oracle)
```

Квантовый оракул выражается матрицей:

$$U_f = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4)$$

В этот момент важно понять, что построение квантового оракула для этого примера уже раскрыло, для какого входа функция возвращает «true». Однако, это только для целей представления, на практике есть квантовый оракул черного ящика, который уже дан алгоритму.

Инициализация. Создадим соединение с виртуальной машиной Quantum, а также инициализируем программу.

```
qvm = QVMConnection()
gr_prog = Program()
```

Во-первых, инициализируем каждый из n = 2 кубитов в состоянии $|0\rangle$. Пакет Rigetti Forest распознает кубиты в обратном порядке, поэтому полезно создать список, содержащий индексы кубитов в порядке убывания.

```
qubits = list(reversed(range(N)))
gr_prog.inst([I(q) for q in qubits])
```

Наше текущее состояние: $|\psi_0\rangle=|00\rangle$

Создание суперпозиции. Следующим шагом является приведение системы в суперпозицию с воротами Адамара:

```
gr_prog.inst([H(q) for q in qubits])
```

Состояние системы после этого шага выражается следующим образом:

$$|\psi_1\rangle = H^{\otimes 2}|\psi_0\rangle = \left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right)\otimes\left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right) = 1/2(|00\rangle+|01\rangle+|10\rangle+|11\rangle) = 1/2\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad (5)$$

Состояние представляет все возможные входы для квантового оракула в одинаково взвешенной суперпозиции. Мы использовали только 2 кубита для представления 4 входных данных.

Петля. Перейдем к той части алгоритма, которая будет повторяться около $\frac{\pi}{4}\sqrt{2^n}$раз. Два шага будут включены в цикл:

1. Применение квантового оракула U_f,
2. Инверсия вокруг среднего.

Квантовый оракул уже задан в качестве входных данных. Осталось добавить его в программу:

ORACLE_GATE_NAME = "GROVER_ORACLE"
gr_prog.defgate(ORACLE_GATE_NAME, oracle)

Что значит инверсия вокруг? Чтобы понять, что значит инверсия вокруг среднего, посмотрим, что произойдет на первой итерации после того, как применим квантовый оракул.

Действие U_f — перевернуть знак амплитуды искомой «string 10»:

$$\left|\psi_2^{(1)}\right\rangle = U_f|\psi_1\rangle = 1/2(|00\rangle+|01\rangle-|10\rangle+|11\rangle) = 1/2\begin{bmatrix}1\\1\\-1\\1\end{bmatrix} \quad (6)$$

Квантового оракула недостаточно, чтобы распознать искомый вход, потому что знак амплитуды не влияет на вероятность измерения. Необходимо искать дополнительные квантовые ворота, которые увеличивают абсолютное значение амплитуды для искомого состояния. Ответ приходит с инверсией вокруг среднего (также называемой диффузионным оператором), которая определяется как $D=2A-I^{\otimes 2}$. А выражается как:

$$A = \begin{bmatrix}1/n & \cdots & 1/n\\ \vdots & \ddots & \vdots\\ 1/n & \cdots & 1/n\end{bmatrix} \quad (7)$$

В нашем примере это:

$$A = 1/2\begin{bmatrix}1&1&1&1\\1&1&1&1\\1&1&1&1\\1&1&1&1\end{bmatrix} \quad (8)$$

и инверсия вокруг среднего имеет вид:

$$D = 2A - I^{\otimes 2} = 1/2\begin{bmatrix}-1&1&1&1\\1&-1&1&1\\1&1&-1&1\\1&1&1&-1\end{bmatrix} \quad (9)$$

Мы можем проверить сами, что это унитарная матрица, то есть действительные квантовые врата. Состояние системы после применения инверсии вокруг среднего значения в первой итерации:

$$\left|\psi_3^{(1)}\right\rangle = D|\psi_2^{(1)}\rangle = 1/4\begin{bmatrix}-1&1&1&1\\1&-1&1&1\\1&1&-1&1\\1&1&1&-1\end{bmatrix}\begin{bmatrix}1\\1\\-1\\1\end{bmatrix} = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \quad (10)$$

У искомой строки амплитуда вероятности равна $\frac{\pi}{4}\sqrt{2^2} \approx 1$ итерации. Это означает, что когда измеряются кубиты, всегда получается искомая «string 10». Однако это определено не является общим случаем для более длинных строк (когда число кубитов больше двух).

Реализуем цикл в Python:

```
# Define quantum oracle
ORACLE_GATE_NAME = "GROVER_ORACLE"
gr_prog.defgate(ORACLE_GATE_NAME, oracle)
# Define inversion around the mean
DIFFUSION_GATE_NAME = "DIFFUSION"
diffusion = 2.0 * np.full((2**N, 2**N), 1/(2**N)) - np.eye(2**N)
```

```
gr_prog.defgate(DIFFUSION_GATE_NAME, diffusion)
# Number of algorithm iterations
N_ITER = int(np.pi / 4 * np.sqrt(2**N))
# Loop
for i in range(N_ITER):
 # \psi_2^i: Apply Quantum Oracle
 gr_prog.inst(tuple([ORACLE_GATE_NAME] + qubits))
 #print(qvm.wavefunction(gr_prog))
 # \psi_3^i: Apply Inversion around the mean
 gr_prog.inst(tuple([DIFFUSION_GATE_NAME] + qubits))
 #print(qvm.wavefunction(gr_prog))
```

Как будет выглядеть ответ сервиса Riggeti Forrest при запуске алгоритма Гровера с некоторыми заданными значениями изображено на рис. 2.

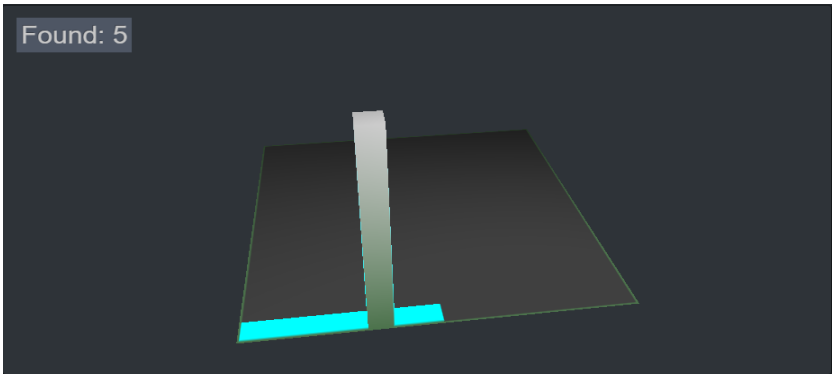

Рис. 2. Графический ответ

Видно, что ответ строится с помощью векторов в трехмерном пространстве. Это помогает наглядно увидеть работу алгоритма.

Измерение. Последний шаг — измерить кубиты и посчитать результат.

```
# \psi_5: Measure
for q in qubits:
 gr_prog.measure(qubit_index=q, classical_reg=q)
# Run
ret = qvm.run(gr_prog, classical_addresses=qubits)
ret_string = ''.join([str(q) for q in ret[0]])
print("The searched string is: {}".format(ret_string))
```

Как уже упоминалось, для случая n = 2 искомая строка измеряется с вероятностью, равной единице. Можно попробовать использовать более длинную строку, например, SEARCHED_STRING = "1011010". В этом сценарии вероятность определения правильного ответа после $\frac{\pi}{4}\sqrt{2^7} \approx 9$ итераций составляет около 99,6%. Это одно из различий между классическим и квантовым компьютером. Многие квантовые алгоритмы возвращают правильный ответ с некоторой вероятностью, тогда как классические компьютеры уверены в результатах вычислений (при условии, что шума нет). Это происходит независимо от квантового шума, то есть неопределенность результатов является неотъемлемым свойством этого квантового алгоритма. Чтобы быть почти на 100% уверенными в ответах от квантовых компьютеров, нужно выполнить измерение несколько раз. Тем не менее, повторение

алгоритма несколько раз не оказывает существенного влияния на квадратичное ускорение при больших n.

Рис. 3. Точность измерений

Заключение. Теоретически алгоритм обеспечивает квадратичное ускорение по сравнению с классическими компьютерами. Это еще не экспоненциальное ускорение, однако оно все еще важно для больших векторов данных. Квантовый параллелизм алгоритма поиска Гровера основывается на одновременном изменении амплитуд всех входов. Это сделано благодаря суперпозиции состояний, которая является чисто квантовой концепцией. Кроме того, поиск выполняется глобально, что указывает на значительное улучшение процедур оптимизации. С другой стороны, алгоритм Гровера чувствителен к количеству итераций. Чем больше итераций, тем меньше будет амплитуда правильного ответа, поэтому неправильный выбор этого параметра может переварить решение. Кроме того, работа алгоритма ограничена в случае введения шума в квантовую систему, которая реальна в современных квантовых компьютерах.

Библиографический список

1. L.K. Grover. A fast quantum mechanical algorithm for database search, Berlin (1996).
2. D. Kopczyk. Quantum machine learning for data scientists, Poland (2018).
3. L. Markov and M. Saeedi, “Constant-optimized quantum circuits for modular multiplication and exponentiation”, Quantum Information and Computation 12, 0361–0394 (2012)
4. Nielson, M. A. and Chuang, I.I., “Quantum Computation and Information”(Cambridge Univ. Press, Cambridge, 2000).
5. Grover L.K.: From Schrödinger's equation to quantum search algorithm, American Journal of Physics, 69(7): 769-777, (2001)
6. Grover's Algorithm: Quantum Database Search, C. Lavor, L.R.U. Manssur, R. Portugal (2015)
7. J. Smith, M. Mosca, "Algorithms for Quantum Computers". (2012)
8. Ambainis, A. "A nearly optimal discrete query quantum algorithm for evaluating NAND formulas". (2017)
9. Farhi Edward, Goldstone Jeffrey, Gutmann Sam. "A Quantum Approximate Optimization Algorithm". (2014)