АННОТАЦИЯ

Разработан приближённый квантовый алгоритм для задачи коммивояжёра. Данный алгоритм относится к вариационным. Он имеет сложность $O(n^2 \log n)$ (n — число городов) и требует $O(n \log n)$ кубитов. Наш алгоритм оказывается производительнее простого перебора и может быть запущен на квантовых компьютерах при текущем уровне развития технологий.

Также был разработан способ представления маршрутов на квантовом компьютере, опирающийся на лексикографическое упорядочивание. Данное представление позволяет автоматически, без введения штрафных слагаемых, учитывать нетривиальные ограничения задачи коммивояжёра, касающиеся того, чтобы маршрут проходил через все города по одному разу и не распадался на не связанные между собой петли.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Квантовый компьютер манипулирует кубитами. Состоящие из кубитов компьютерные регистры могут находиться в суперпозиции нескольких состояний. Когда квантовый компьютер производит вычисление над суперпозицией, он производит вычисление над всеми состояниями одновременно [1]. Это делает квантовые компьютеры привлекательными для задач, в которых требуется перебор большого числа вариантов.

В этой работе мы попытаемся решить задачу коммивояжёра.

Будем искать решение в классе вариационных квантовых алгоритмов [2]. В них квантовый компьютер запускается снова и снова с небольшими изменениями в параметрах квантовой цепи. Процессом варьирования управляет классический компьютер. Его задача — подобрать параметры цепи такие, чтобы на выходе получалось оптимальное по некоторому заданному критерию состояние.

Уже разработаны вариационные квантовые алгоритмы для задач из нескольких разных областей. Среди них — алгоритмы поиска собственных значений матрицы [3], максимального разреза графа [4] и другие. Они позволяют решать задачи на существующих квантовых компьютерах — подверженных шумам и оперирующих малым числом кубитов.

Для задачи коммивояжёра, которой посвящена данная работа, эффективного квантового алгоритма пока не существует.

Особенностью этой задачи является наличие ограничений: например, коммивояжёр не может посетить один город дважды. Традиционно ограничения в вариационных квантовых алгоритмах реализуются через дополнительные штрафные слагаемые к оптимизируемой величине [5, 6, 7]. Но мы вместо этого попробуем добиться того, чтобы состояние, не удовлетворяющее ограничениям, в принципе невозможно было измерить.

Такая реализация ограничений задачи непосредственно через саму квантовую цепь — главное направление этой работы.

Настоящие квантовые компьютеры всё ещё не широко доступны, поэтому мы тестируем алгоритм с помощью компьютерной симуляции. Для этого мы используем `cirq` — библиотеку для Python от Google Quantum AI [8].

1 Классические подходы

1.1 Постановка задачи коммивояжёра

В задаче коммивояжёра есть n городов, соединённых друг с другом, и требуется найти кратчайший маршрут, проходящий через все города по одному разу.

Существует несколько вариаций задачи. Маршрут может быть замкнутым или незамкнутым, расстояния между городами могут подчиняться или не подчиняться неравенству треугольника; граф городов может быть полным или неполным, ориентированным или неориентированным. Все эти разновидности можно свести друг к другу, поэтому мы можем выбрать ту формулировку, которая кажется наиболее удобной.

Итак, пусть каждый город соединён с каждым (т.е. граф городов полный), пусть мы ищем незамкнутый маршрут и пусть граф городов ориентированный. Ориентированность означает, что условная длина (или стоимость) пути из города A в город B необязательно равна стоимости пути из B в A.

Введём обозначение

$$l_{ij} - \text{ длина пути из } i\text{-того города в } j\text{-тый}$$

Ответом на задачу служит последовательность городов. Есть n способов выбрать первый город, $n-1$ способ выбрать второй и так далее — всего $n!$ возможных ответов. Если бы мы выбрали другую формулировку задачи, то пришлось бы, например, учитывать, что в замкнутом пути начальную точку можно выбрать несколькими способами.

Простейший алгоритм — перебрать все возможные перестановки — будет иметь сложность $O(n!)$ по времени и $O(1)$ по памяти.

1.2 Алгоритм Хелда–Карпа

Альтернативой является алгоритм Хелда–Карпа, который имеет сложность $O(n^2 2^n)$ по времени и $O(n2^n)$ по памяти. Это лучший классический алгоритм из тех, которые гарантированно приходят к точному решению.[1] Рассмотрим его подробнее, чтобы лучше понять, с чем мы соревнуемся.

[1] Ещё есть эвристические приближённые алгоритмы

Для нашей постановки задачи алгоритм выглядит следующим образом:

1) Создаётся мнимый нулевой город, связанный со всеми остальными городами дорогами длины 0.

2) Определяется рекурсивная функция $g(S, e)$, возвращающая длину наименьшего пути, начинающегося в городе 0, проходящего через каждый город некоторого подмножества городов S и заканчивающегося в городе $e \notin S$.

3) Эта функция вычисляется следующим образом:

$$g(S, e) = \min_{e' \in S} \left\{ g(S \backslash e',\ e') + l_{e'e} \right\}$$

(через перебор всех возможных вариантов для предпоследнего города e')

4) Существует 2^n возможных множеств S. Если последовательно вычислять $g(S, e)$ от меньших множеств к большим, перебирая все $e \notin S$, то мы сможем использовать уже вычисленные значения g, и «входить в рекурсию» не потребуется. Функция g будет вызвана $O(n2^n)$ раз, и на хранение результатов потребуется $O(n2^n)$ памяти.

5) При каждом вызове функции $g(S, e)$ происходит перебор $O(n)$ возможных предпоследних вершин e', поэтому сложность функции g — $O(n)$, а сложность алгоритма в целом — $O(n^2 2^n)$.

6) В конце вычисляется $g(S_{\text{all}}, 0)$, где S_{all} — множество всех городов. Это и будет длиной искомого минимального пути.

7) Можно восстановить и сам оптимальный путь, а не только его длину, если в ходе решения помимо промежуточных значений функции $g(S, e)$ сохранять номера оптимальных предпоследних вершин e'. Это не повлияет на асимптотику алгоритма.

Алгоритм Хелда–Карпа работает быстрее, чем простой перебор, но требует экспоненциально много памяти.

2 Обзор литературы

2.1 Решение на основе алгоритма Гровера

Один из способов решить задачу коммивояжёра на квантовом компьютере — это использовать алгоритм Гровера. Это известный квантовый алгоритм, который позволяет «отфильтровать» состояния, удовлетворяющие какому-либо критерию. При этом достигается квадратичное ускорение по сравнению с полным перебором: если всего состояний N, а k из них удовлетворяют критерию, то время работы алгоритма составит $O\left(\sqrt{N/k}\right)$.

На основе алгоритма Гровера можно создать алгоритм поиска минимума дискретной функции, которую в данной работе мы будем называть стоимостью.

В самом деле, мы можем выбрать какое-то произвольное решение, отфильтровать все решения со стоимостью меньше, чем у выбранного, и повторить процесс. В работе Durr, Hoyer [9] показано, что мы придём к минимуму за $O\left(\sqrt{N}\right)$ итераций с вероятностью не меньше 50%.

Осталось применить квантовый алгоритм поиска минимума к задаче коммивояжёра, для этого нужно разработать квантовую схему, которая будет вычислять стоимость маршрута и сравнивать её с пороговой.

В 2018 году Srinivasan et al. [10] предложили вычислять стоимость маршрута, закодировав стоимости индивидуальных рёбер в виде сдвигов фаз $e^{il_{ij}}$ так, чтобы суммарный набег фазы у квантового состояния $e^{il_{i_1j_1}} \cdot e^{il_{i_2j_2}} \cdots = e^{iL}$ был пропорционален суммарной стоимости маршрута. Затем авторы применяют квантовый алгоритм оценки фазы, родственный преобразованию Фурье.

А схему для сравнения стоимости с пороговой разработали Gilliam et al. в 2021 году [11]. Они предложили вычислять разность и отслеживать бит знака. Вычитание они реализовали через квантовое преобразование Фурье.

Главной проблемой описанного решения на основе алгоритма Гровера является то, что его сложность $O\left(\sqrt{N}\right) = O\left(\sqrt{n!}\right)$ хуже, чем сложность классического алгоритма Хелда–Карпа $O(n^2 2^n)$ (рис. 2.1).

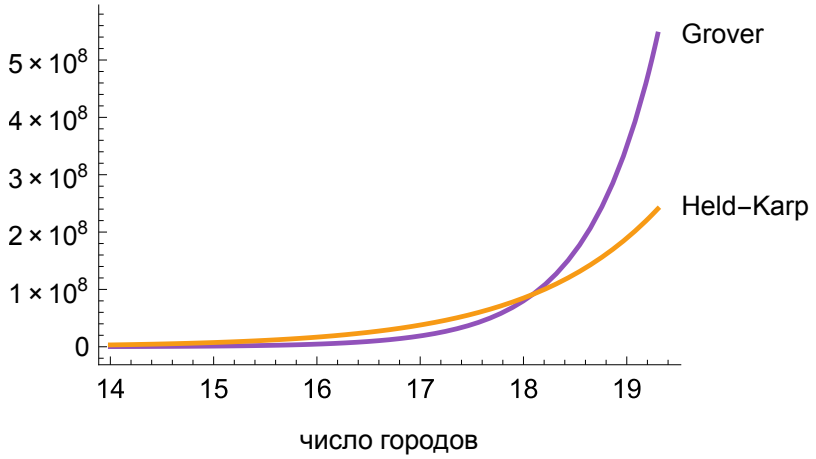


Рисунок 2.1 — Сложность квантового алгоритма решения задачи коммивояжёра на основе алгоритма Гровера $\left(O(\sqrt{n!})\right)$, и сложность классического алгоритма Хелда–Карпа $O(n^2 2^n)$.

2.2 Квантовый алгоритм Хелда–Карпа

Классический алгоритм Хелда–Карпа оказался эффективнее квантово ускоренного простого перебора. Возникает вопрос: можно ли тогда на квантовом компьютере ускорить сам алгоритм Хелда–Карпа?

В работе [12] приведён такой алгоритм. Идея авторов заключается в том, чтобы сначала найти решения для части подмножеств городов, а затем использовать алгоритм Гровера и квантовый алгоритм поиска минимума на остальных подмножествах, чтобы найти ответ на задачу. В результате получается сложность $O^*(1.728^n)$.

Другой алгоритм приведён в работе [13]. Авторы рассматривали только графы с ограниченной степенью, это позволило им дополнительно ускорить алгоритм Хелда–Карпа, используя квантовый аналог поиска с возвратом. Им удалось получить сложность $O^*(1.110^n)$ для графов степени 3; $O^*(1.301^n)$ для графов степени 4; $O^*(1.680^n)$ для графов степени 5 и 6.

Основной проблемой этих алгоритмов (помимо их экспоненциальной сложности), является то, что им требуется доступ к квантовой памяти. К сожалению, на данный момент реализаций квантовой памяти не существует.

2.3 Квантовый отжиг

D-Wave Systems, единственная коммерчески успешная квантовая компания, продаёт устройства, предназначенные для квантового отжига. Эти устройства производят адиабатические квантовые вычисления и способны

решать задачу квадратичной бинарной оптимизации без ограничений (QUBO). Другие задачи с их помощью решить нельзя, поэтому задачу коммивояжёра имеет смысл переформулировать как QUBO.

Задача QUBO формулируется следующим образом: нужно найти минимум функции

$$f(\vec{x}) = \sum_i a_i x_i + \sum_{i,j} c_{ij} x_i x_j$$

где $x_i \in \{0, 1\}$ — бинарные переменные, a_i и c_{ij} — постоянные коэффициенты. Линейные члены $a_i x_i$ можно не писать отдельно, если заметить, что для бинарных переменных $x_i = x_i^2$.

Один из простейших способов [14] переформулировать задачу коммивояжёра как QUBO — это ввести бинарные переменные

$$x_{it} = \begin{cases} 1, & \text{если на шаге } t \text{ посещён город } i \\ 0 & \text{в остальных случаях} \end{cases}$$

В таком случае стоимость пути будет равна

$$C = \sum_{i,j,t} l_{ij} x_{i,t} x_{j,t+1}$$

Должно быть выполнено ограничение $\sum\limits_i x_{it} = 1 \ \forall t$ (в каждый момент времени посещён один город) и $\sum\limits_t x_{it} = 1 \ \forall i$ (каждый город посещён один раз). Чтобы обеспечить их выполнение, к стоимости пути добавляют штрафные слагаемые:

$$\tilde{C} = C + p \cdot \left(1 - \sum_i x_{it}\right)^2 + p \cdot \left(1 - \sum_t x_{it}\right)^2,$$

Константа p должна быть достаточно большой, чтобы нельзя было получить преимущество, нарушив ограничения. В частности, должно быть выполнено

$$p > \max_{ij} l_{ij}$$

Другой способ выбрать бинарные переменные приведён в работе [5]:

$$x_{u,v}^{t} = \begin{cases} 1, & \text{если на шаге } t \text{ коммивояжёр перемещается из города } u \text{ в город } v \\ 0 & \text{в остальных случаях} \end{cases}$$

В таком случае необходимо отслеживать, чтобы маршрут не распадался на не связанные между собой петли. Для этого необходимо, чтобы город, в который коммивояжёр прибыл на шаге t, и город, который он покинул на шаге $t + 1$, совпадали. Авторы работы [5] (2022) выводят соответствующий громоздкий штрафной член.

Недостатком алгоритмов квантового отжига (помимо наличия штрафных членов) является то, что время работы быстро растёт с увеличением размера задачи и уменьшением промежутков между решениями. В типичных задачах время работы оказывается экспоненциальным [14]:

$$T = O\left(\exp(\alpha n^{\beta})\right)$$

Остаётся открытым вопрос, является ли квантовый отжиг хоть немного более эффективным, чем классические алгоритмы.

В данной работе мы не будем ограничиваться QUBO-формулировкой задачи коммивояжёра и придумаем своё кодирование. Это позволит избежать введения штрафных членов и уменьшить количество кубитов.

2.4 Quantum Approximate Optimization Algorithm

Quantum Approximate Optimization Algorithm (QAOA) [15] — вариационный алгоритм, первоначально предложенный для задачи о максимальном разрезе графа. QAOA — это единственный вариационный квантовый алгоритм, для которого доказано, что в пределе бесконечно глубокой цепи с его помощью можно получить точное решение.

Идея алгоритма заключается в том, чтобы выбрать два гамильтониана: перемешивающий гамильтониан $\hat{B}$ и гамильтониан оптимизируемой величины $\hat{C}$. Пусть начальное состояние системы было основным для гамильтониана $\hat{B}$. Тогда при эволюции под воздействием гамильтониана

$$\hat{H}(t) = (1 - t/T) \cdot \hat{B} + (t/T) \cdot \hat{C} \tag{2.1}$$

система перейдёт в основное состояние гамильтониана $\hat{C}$ по адиабатической теореме из квантовой механики, если время T достаточно большое.

Однако реализовать непрерывно меняющийся гамильтониан (2.1) на квантовом компьютере может быть сложно, поэтому в алгоритме QAOA используют аппроксимацию, в которой эволюция под воздействием $\hat{B}$ и $\hat{C}$ происходит поочерёдно:

$$\hat{U} = \ldots e^{-i\beta_2\hat{B}} e^{-i\gamma_2\hat{C}} \cdot e^{-i\beta_1\hat{B}} e^{-i\gamma_1\hat{C}}$$

Промежутки времени γ_i и β_i являются вариационными параметрами, которые необходимо подобрать путём многократных запусков.

Обычно в качестве перемешивающего гамильтониана берут сумму однокубитных операторов $\hat{X} = \hat{\sigma}_x = \mathrm{NOT}$:

$$\hat{B} = \bigoplus_i \hat{X}_i \qquad e^{-i\beta\hat{B}} = \bigotimes_i \mathrm{Rx}(\beta)$$

а в качестве начального состояния — равную суперпозицию всех состояний:

$$|\psi_0\rangle = \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)^{\otimes m}$$

Один из способов применить алгоритм QAOA к задаче коммивояжёра — это переформулировать её как задачу QUBO со штрафными слагаемыми подобно тому, как описано в пункте «Квантовый отжиг». Такой подход применяется в работах [16, 17] и на домашней странице Джека Церони [18].

Важным свойством QUBO-формулировки является то, что гамильтониан $\hat{C}$ оказывается записан в виде суммы коммутирующих слагаемых:

$$\hat{C} = \sum_{i,j} c_{ij} \cdot \hat{q}_i \otimes \hat{q}_j \qquad \hat{q} = |1\rangle\langle 1|,$$

где индексы означают номер кубита. Поскольку для коммутирующих операторов $e^{\hat{x}+\hat{y}} = e^{\hat{x}} \cdot e^{\hat{y}}$, это позволяет выразить оператор эволюции $e^{-i\gamma\hat{C}}$ через операторы эволюции для отдельных слагаемых. Эти операторы являются двухкубитными и их можно реализовать на квантовом компьютере; они выражаются через гейты CNOT и Rz.

Однако, такие алгоритмы требуют $O(n^2)$ кубитов. Ни в одной из цити-

руемых работ не удалось промоделировать графы с числом городов больше четырёх.

2.5 Quantum Alternating Operator Ansatz

Quantum Alternating Operator Ansatz [19] (2019) является расширением Quantum Approximate Optimization Algorithm, оба алгоритма сокращаются как QAOA.

В расширенной версии алгоритма предлагается реализовать ограничения задачи следующим образом. Начальным состоянием должна быть не равная суперпозиция всех 2^m состояний, а какое-либо одно допустимое решение. Далее, как и в старом алгоритме, поочерёдно применяются операторы $e^{-i\beta_i \hat{B}}$ и $e^{-i\gamma_i \hat{C}}$. Разница заключается в том, что теперь оператор $\hat{B}$ должен переводить допустимые решения в допустимые. В случае задачи коммивояжёра авторы работы [19] предлагают использовать в качестве оператора $\hat{B}$ различные комбинации SWAP-гейтов.

Авторы работы [20] (2020) идут более сложным путём и конструируют квантовый оракул, который определяет, является ли решение допустимым маршрутом. Затем они конструируют с его помощью перемешивающий гамильтониан $\hat{B}$ в виде

$$\hat{B} = \sum_{|x\rangle,|x'\rangle \in F} \Big(|x'\rangle\langle x| + |x\rangle\langle x'| \Big),$$

где F — множество допустимых решений.

Такой алгоритм позволяет избавиться от штрафных слагаемых в гамильтониане, так как состояния, не удовлетворяющие ограничениям, просто не возникают. Возможно, именно по этому пути пойдёт развитие квантовых технологий. Тем не менее в данном дипломе мы попробовали разработать с нуля более простое решение, не требующее оракулов и использующее меньшее количество кубитов ($O(n \log n)$ вместо $O(n^2)$ в работе [20]).

3 Описание алгоритма

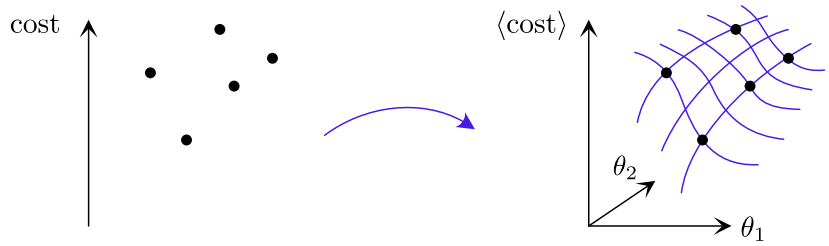
На рисунке 3.1 приведена общая схема нашего алгоритма. В начале некоторыми практически произвольными параметрическими гейтами создаётся суперпозиция нескольких возможных маршрутов. Затем кубиты измеряются. Это происходит несколько раз, и на классическом компьютере вычисляется средняя стоимость маршрута.

Рисунок 3.1 — Общая схема нашего алгоритма

Основываясь на средней стоимости, мы обновляем параметрические гейты и запускаем цепь заново в надежде, что средняя стоимость уменьшится. Здесь мы можем следовать любому численному алгоритму нахождения минимума функции. Лучше всего себя показал алгоритм Rotosolve [21] (2021), который опирается на то, что все квантовые гейты — унитарные, и, следовательно, функция $\langle \text{cost} \rangle = f(\theta_1, \theta_2 \dots)$ (зависимость средней стоимости от параметров гейтов) не совсем произвольная.

Квантовый компьютер нужен для того, чтобы преобразовать пространство поиска. Исходная задача была дискретной, но, добавив «прослойку» в виде квантового компьютера, мы преобразовали её к непрерывной (рис. 3.2). Непрерывность появляется за счёт того, что на квантовом компьютере мы можем «смешивать» несколько маршрутов в суперпозицию, и стоимость получившейся суперпозиции будет находиться где-то между дискретными значениями исходных маршрутов. Можно показать, что задача в результате получается не только непрерывной, но и гладкой. Это позволяет применять всё многообразие градиентных и неградиентных методов оптимизации, в том числе выбранный нами алгоритм Rotosolve.

Рассмотрим теперь детали нашего решения.



дискретная задача: непонятно, как оптимизировать

непрерывная задача: можем использовать градиентный спуск, алгоритм Rotosolve и т.д.

Рисунок 3.2 — Введение квантового компьютера позволяет преобразовать дискретную задачу к непрерывной.

3.1 Представление маршрутов на квантовом компьютере

Коротко: все возможные маршруты можно занумеровать в лексикографическом порядке. Этот номер кодируется на квантовом компьютере, для чего требуется $\lceil \log_2 n! \rceil$ кубитов. В конце схемы кубиты измеряются, и на классическом компьютере номер маршрута переводится в факториальную систему счисления, затем вычисляется сам маршрут и его стоимость.

Рассмотрим эту процедуру подробнее.

3.1.1 Маршруты как перестановки

Мы выбрали вариант задачи коммивояжёра, в котором требуется найти незамкнутый маршрут в полном ориентированном графе. Корректным маршрутом является любая последовательность посещённых городов, в которой каждый город встречается по одному разу — то есть любая из $n!$ перестановок городов. В нашей формулировке задачи между маршрутами и перестановками есть взаимно однозначное соответствие; мы можем использовать эти слова как синонимы.

Присвоим городам номера от 0 до $n - 1$.

Перестановки можно строить с помощью дерева (рис. 3.3): в начале мы выбираем первый город из n вариантов, затем с каждым следующим городом количество ветвей уменьшается на 1. Ветви у каждого узла расположены по возрастанию. Перестановки, соответствующие листьям дерева, будут расположены тогда в лексикографическом порядке.

Поскольку перестановки теперь упорядочены, мы можем их пронумеровать. Номер перестановки удобно записывать в факториальной системе счисления, поскольку эта система отражает структуру дерева вариантов.

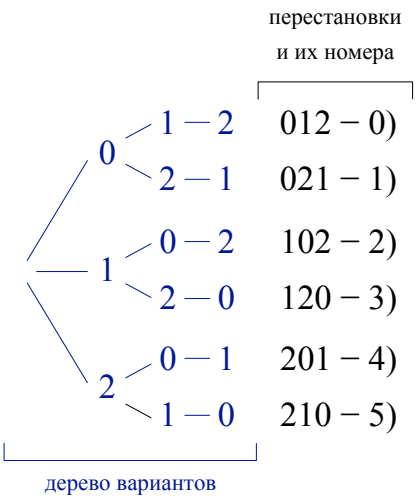


Рисунок 3.3 — Все возможные маршруты для случая трёх городов в лексикографическом (т.е. «алфавитном») порядке.

3.1.2 Факториальная система счисления

Обсудим факториальную систему счисления. В этой системе последняя цифра должна быть из множества $\{0, 1\}$, вторая с конца — из $\{0, 1, 2\}$, третья — из $\{0, 1, 2, 3\}$ и так далее. Если в десятичной системе цифры умножаются на числа 1, 10, 10^2 и так далее, то в факториальной — на числа 1!, 2!, 3!. . . :

$$\overline{\ldots cba}_! = a \cdot 1! + b \cdot 2! + c \cdot 3! \ldots$$

Максимальное число из $n - 1$ цифр равно $n! - 1$.

Если записать номер перестановки в факториальной системе счисления, то последовательность его цифр можно будет интерпретировать как последовательность выборов в дереве вариантов с рисунка 3.3. В самом деле, первая цифра в $n-1-$значном числе соответствует выбору из n вариантов, вторая — выбору из $n - 1$ варианта, последняя — выбору из двух вариантов. Взяв число

$$a = \overline{a_1 a_2 \ldots a_{n-1}}_!$$

двигаясь по дереву и выбирая на k-том шаге ветвь номер a_k, мы придём в конце к перестановке номер a.

Таким образом, имея номер перестановки в факториальной системе счисления, мы можем цифра за цифрой восстановить по нему саму перестановку. Хранить факториально большое дерево для этого не нужно: доступные

ветви на каждом шаге определяются ещё не использованными городами.

3.1.3 Процедура декодирования маршрутов

Вышесказанное позволяет в качестве представления маршрута использовать его номер $c \in [0, n!)$. Любое целое число, попадающее в этот промежуток, является валидным маршрутом.

Такое представление выгодно отличается от представления маршрута через его рёбра, которое использовалось в других работах (см. раздел «Обзор литературы»). Наше представление автоматически учитывает нетривиальные ограничения задачи, касающиеся того, чтобы маршрут проходил через все города и не распадался на несвязанные друг с другом петли.

На квантовом компьютере для хранения номера маршрута мы будем использовать двоичную систему счисления, что потребует $\lceil \log_2 n! \rceil = O(n \log n)$ кубитов.

Хотелось бы построить такую схему, которая генерирует только валидные номера перестановок — то есть из промежутка $[0, n!)$. Однако мы не будем этим себя утруждать и позволим схеме генерировать все номера $b \in [0,\ 2^{\lceil \log_2 n! \rceil})$, которые могут быть записаны на имеющемся наборе кубитов.

Слишком большие номера перестановок мы реинтерпретируем на классическом компьютере как номера из допустимого множества по формуле

$$c = b \ \% \ n!$$

После измерения мы переводим номер перестановки из двоичной системы в факториальную, используя обычный алгоритм перевода между системами счисления, основанный на делении с остатком.

Затем на классическом компьютере мы вычисляем сам маршрут, а потом, зная маршрут, вычисляем его стоимость. Этот процесс повторяется достаточное количество раз, чтобы вычислить среднюю стоимость состояния, которое в общем случае представляет собой суперпозицию разных маршрутов.

3.2 Квантовая цепь

На рис. 3.4 изображена использованная нами квантовая схема. Она состоит из слоя гейтов Rx, за которым идёт слой гейтов CNOT. В схему можно

добавить ещё несколько таких пар слоёв, но мы не стали этого делать. Гейты поворота Rx имеют параметры: углы θ_i. Обсудим, почему мы выбрали эту схему.

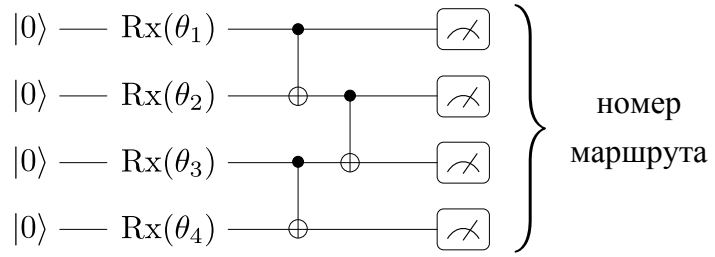


Рисунок 3.4 — Квантовая схема, использованная в разработанном алгоритме

Нам было нужно параметрически создавать квантовое состояние, которое соответствует какому-то маршруту или их суперпозиции. Исходя из этого, мы выдвинули три требования к квантовой цепи:

1) все состояния-маршруты должны быть «достижимыми»: для каждого маршрута должен существовать набор параметров такой, что на выходе цепи должно получиться состояние, соответствующее этому маршруту

2) все достижимые состояния должны быть «корректными»: при любом наборе параметров на выходе цепи должна получаться суперпозиция допустимых маршрутов

3) параметрические гейты должны позволять использовать алгоритм Rotosolve, который мы обсудим в следующем разделе. Он применим к широкому классу гейтов, но не ко всем.

Ранее мы выбрали способ представления маршрутов (через их номер), в котором все состояния, составляющие вычислительный базис, соответствуют какому-то допустимому маршруту. А совершенно произвольное состояние, таким образом, соответствует некоторой суперпозиции допустимых маршрутов, и требование (2) выполнено автоматически.

Требование достижимости (1) тоже выполнено. В этом несложно убедиться, если сначала заметить, что для создания произвольного маршрута (двоичного числа, символизирующего его номер) было бы достаточно одного слоя гейтов Rx. Углы в остальных слоях (если они есть) можно положить равными 0, тогда останутся только гейты CNOT, которые легко обращаются.

Чтобы выполнить все ограничения, гейты CNOT не понадобились. Одна-

ко они устанавливают связь между кубитами, поэтому мы их добавили. Это улучшило результат в симуляции.

Обсудим также то, как кубиты должны быть соединены между собой на физическом устройстве. Эта тема остаётся «больным местом» физических реализаций квантовых компьютеров. Никому до сих пор не удалось создать квантовый компьютер, в котором каждый кубит был бы связан с каждым, и это вряд ли изменится когда-либо в обозримом будущем.

К счастью, в нашей схеме двухкубитные гейты производятся только над соседними кубитами, следовательно, кубитам достаточно просто быть соединёнными в линию (рис. 3.5).

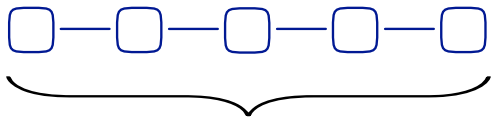


Рисунок 3.5 — Required qubit connectivity

Квантовые компьютеры с такой простой структурой можно ожидать в ближайшем будущем.

3.3 Алгоритм Rotosolve для подбора параметров

Наша схема содержит гейты поворота $\mathrm{Rx}(\theta)$, и в ходе алгоритма мы пытаемся подобрать оптимальные углы θ, минимизирующие среднюю стоимость состояния, получающегося на выходе. Для этого мы могли бы использовать любой алгоритм поиска минимума функции, например метод градиентного спуска.

Но в 2021 году Ostaszewski et al. [21] предложили более эффективный алгоритм, названный Rotosolve. Они заметили, что большинство доступных квантовому компьютеру параметрических гейтов может быть представлено в виде

$$\hat{U}(\theta) = \exp\left(-\frac{i\theta}{2}\hat{H}\right) = \cos\left(\frac{\theta}{2}\right)\hat{1} - i\sin\left(\frac{\theta}{2}\right)\hat{H}, \tag{3.1}$$

где $\hat{H}$ — это некоторая унитарная эрмитова матрица — такая, что $\hat{H}^2 = \hat{1}$.

В частности, при $\hat{H} = \hat{\sigma}_x$, $\hat{\sigma}_y$, $\hat{\sigma}_z$ мы получаем знакомые гейты поворота Rx, Ry, Rz. Физический смысл параметра θ заключается во времени воздействия на кубит.

Из формулы (3.1) видно, что если зафиксировать параметры всех гейтов, кроме одного, то математическое ожидание произвольной наблюдаемой C будет иметь синусоидальную форму:

$$\langle C \rangle_\theta = a \sin(\theta + b) + c$$

Если мы сможем оценить коэффициенты a, b и c, то мы сможем охарактеризовать синусоиду и найти минимум. Авторы показали, что он с точностью до $2\pi k$ даётся выражением в замкнутой форме

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \langle C \rangle_\theta = \\ = \theta_0 - \frac{\pi}{2} - \operatorname{arctan2}\left(2\langle C \rangle_{\theta_0} - \langle C \rangle_{\theta_0 + \frac{\pi}{2}} - \langle C \rangle_{\theta_0 - \frac{\pi}{2}}, \quad \langle C \rangle_{\theta_0 + \frac{\pi}{2}} - \langle C \rangle_{\theta_0 + \frac{\pi}{2}}\right),$$

где θ_0 — произвольная начальная точка.

Алгоритм Rotosolve оптимизирует углы для всех гейтов по очереди, затем цикл повторяется, пока не выполнится критерий остановки.

3.4 Критерий остановки

Выбор критерия остановки нетривиален, поскольку функция, которую мы минимизируем, зашумлена из-за взаимодействия кубитов со средой и из-за случайности самого акта измерения.

В ходе алгоритма значения параметров обновляются циклически. Обозначим $c[i][j]$ — значения целевой функции на i-том цикле перед обновлением j-того параметра.

В качестве критерия остановки можно использовать:

0) ограничение на максимальное число итераций

1) $\left|c[i-1][0] - c[i][0]\right| < tol$

2) $c[i-1][0] - c[i][0] < tol$

3) $\left\|(\vec{\theta}[i] - \vec{\theta}[i-1] + \pi) \ \% \ 2\pi - \pi\right\| < tol,$

где $\vec{\theta}[i]$ — вектор параметров перед i-тым циклом обновления, tol — фиксированное маленькое число.

Из-за шумов значение функции на последующем шаге может быть больше, чем на предыдущем, поэтому критерии (1) и (2) не эквивалентны.

Вслед за разработчиками библиотеки TensorFlow Quantum [22] мы используем комбинацию вариантов (0) и (1). Но другие критерии тоже возможны и имеют свои преимущества. Например, критерий (3) позволяет выбрать величину *tol* не зависящей от характерного масштаба значений функции.

То, как на практике работают данные критерии остановки, иллюстрирует рис. 3.6:

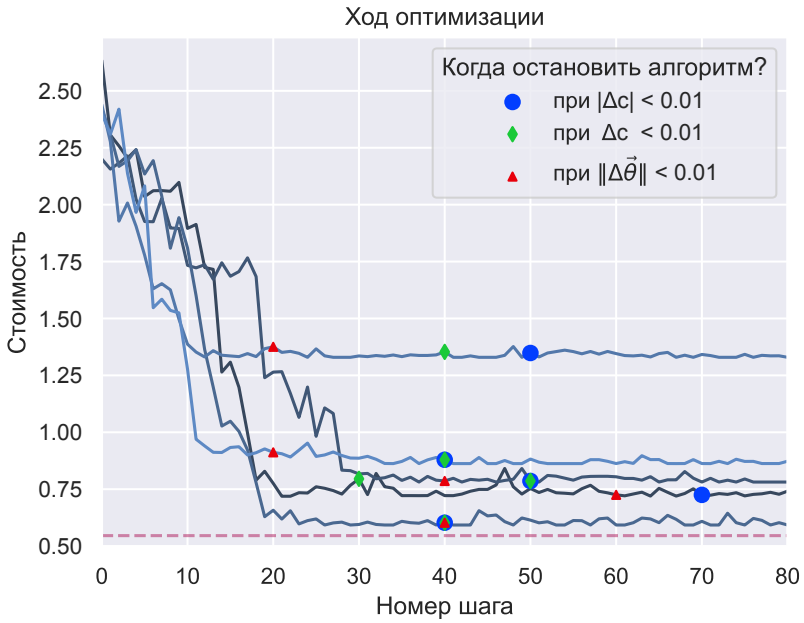


Рисунок 3.6 — Сплошные линии показывают ход оптимизации при старте из различных случайных начальных точек. Пунктирной линии соответствует точное решение. Отмечены точки, в которых сработали бы критерии остановки (1)–(3). Эти критерии могут остановить алгоритм только между циклами, каждый из которых в данном случае (6 городов) состоит из 10 шагов. Для дальнейшего использования мы выбрали критерий (1): $|\Delta c| < 0.01$.

Неправильно выбранный критерий остановки может заставить алгоритм совершать лишние итерации или наоборот может заставить остановить итерации слишком рано.

Отметим, что не обязательно в условии остановки использовать только две точки. Так, например, в книге [23] предлагается использовать критерии, основанные на скользящем среднем. Сравнение разных критериев остановки может быть стоящим направлением будущих исследований, как в применении к нашей задаче, так и для вариационных квантовых алгоритмов в целом.

4 Оценки эффективности алгоритма

Мы протестировали наш алгоритм на случайных графах с 4–10 городами (от 24 до 3 628 800 возможных решений). Стоимости путей между всеми парами городов генерировались равномерно на отрезке [0, 1]. Мы рассмотрели самую общую задачу, когда стоимость пути может быть разной в зависимости от направления.

4.1 Качество решений

Для начала рассмотрим случай 4 городов. Мы построили распределения стоимостей на входе и на выходе алгоритма (рис. 4.1). Вообще говоря, алгоритм Rotosolve детерминистический: каждая следующая точка в пространстве параметров однозначно определяется предыдущей. Однако мы, во-первых, запускали его из различных случайных начальных точек, и во-вторых, оценивали матожидание стоимости с помощью сэмплирования, что вносит случайную ошибку. Из-за этого, запуская алгоритм несколько раз на одном графе, мы получали разные результаты. Их распределение и изображено на рис. 4.1.

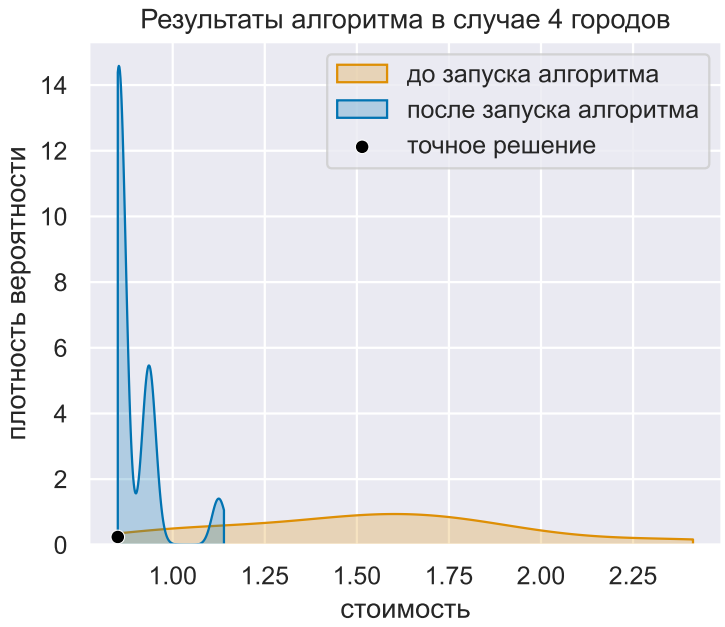


Рисунок 4.1 — Оранжевое: распределение стоимостей всех возможных $n!$ ($n = 4$) маршрутов. Синее: распределение стоимостей решений, к которым приходит алгоритм при старте из различных случайных начальных точек. Чёрной точкой на оси X отмечена стоимость точного решения. В большинстве случаев решение совпадало с точным, это соответствует высокому левому пику. Он имеет ненулевую ширину из-за того, что при построении графика использовался метод ядерной оценки плотности.

Из рисунка 4.1 видно, что в случае 4 городов алгоритм практически всегда выдаёт точное решение. К сожалению, при большем числе городов такого чуда больше не происходит. Однако алгоритм продолжает стабильно выдавать решения с маленькой стоимостью.

Чтобы убедиться в этом, мы построили аналогичные распределения при другом числе городов и объединили их в одну скрипичную диаграмму (рис. 4.2). Для удобства сопоставления мы отнормировали стоимость на среднее по всем перестановкам. Это среднее приблизительно равно $\frac{n-1}{2}$, поскольку стоимость каждого ребра равномерно распределена на [0, 1], а таких рёбер в незамкнутом маршруте ровно $n - 1$.

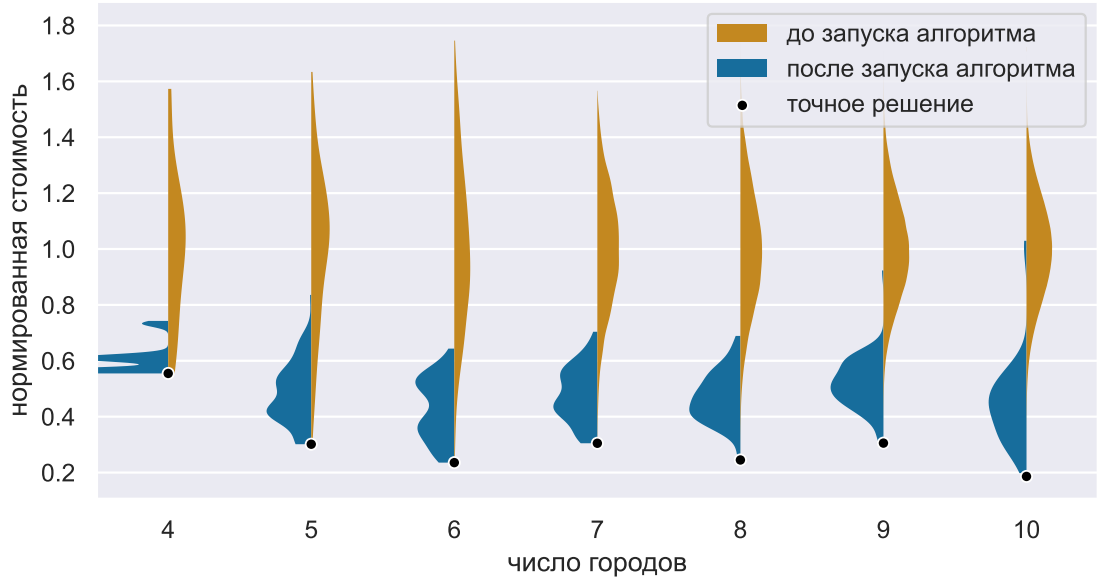


Рисунок 4.2 — Распределения стоимостей до и после оптимизации в задачах с разным числом городов.

Из рисунка 4.2, во-первых, видно, что оранжевое распределение при росте числа городов в силу центральной предельной теоремы стремится к нормальному. Во-вторых, синее распределение не смещается вверх, это означает, что алгоритм продолжает выдавать решения одного качества при любом числе городов, их стоимость составляет около 0.5 от средней.

Наш алгоритм может быть полезен на практике, когда нужно за маленькое количество итераций найти решение с небольшой стоимостью (необязательно точное). Кроме того, полученное приближённое решение можно использовать в качестве стартовой точки для классического алгоритма, например, для локального поиска или генетических алгоритмов.

4.2 Сложность

Вычислительная сложность — едва ли не самая главная характеристика алгоритмов.

В данном алгоритме параметры обновляются циклически, каждое обновление параметра требует 3 вычисления функции. Обозначим количество циклов как K, а количество параметров как $l = \lceil \log_2 n! \rceil = O(n \log n)$. Тогда сложность алгоритма равна

$$O(K \cdot n \log n)$$

Чтобы оценить количество циклов, которое понадобится алгоритму, чтобы сойтись, мы построили следующий график (рис. 4.3):

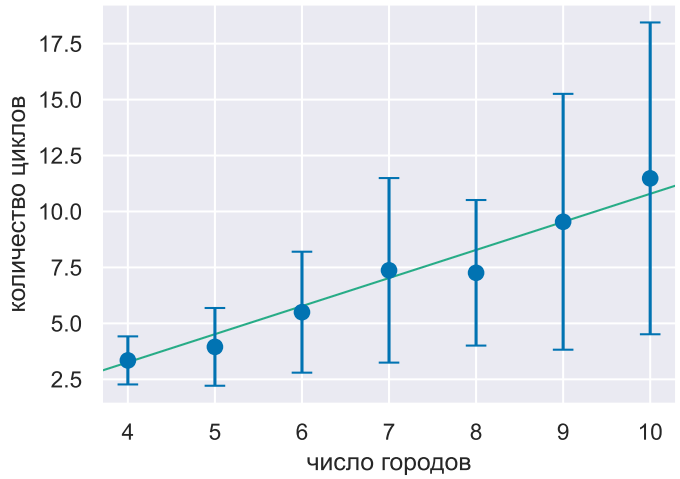


Рисунок 4.3 — Зависимость количества циклов K обновления параметров от числа городов. Планки означают стандартное отклонение, точки означают среднее по большому количеству запусков с разными случайными начальными точками.

Как видно из графика, количество циклов приблизительно линейно, то есть $K = O(n)$. Окончательно, сложность нашего алгоритма равна

$$O(n^2 \log n) \quad \text{вычислений стоимости,}$$

что значительно лучше классических алгоритмов ($n!$ для простого перебора и $n^2 2^n$ у алгоритма Хелда–Карпа). Однако наш алгоритм выдаёт не совсем оптимальные решения, поэтому для сравнения алгоритмов понадобились провести более полный анализ.

4.3 Сравнение с простым перебором

Предположим, мы запустили наш алгоритм на некоторых входных данных, и после M итераций получили решение со стоимостью c. Хорошо это или плохо?

Введём величину $p(c)$ — долю решений со стоимостью, меньшей или равной c (процентильный ранг). Заметим, что в случае простого перебора

$$M \cdot p(c) \approx 1,$$

где $M = nfev$ — количество вычислений целевой функции. Например, чтобы получить решение из лучших 50%, в среднем достаточно двух попыток, а чтобы получить наилучшее решение $\big(p(c) = 1/n!\big)$, требуется перебрать $n!$ кандидатов.

Мы можем вычислить величину $M \cdot p(c)$ для нашего алгоритма и сравнить с единицей (рис. 4.4). Если она меньше единицы, то алгоритм лучше простого перебора, если больше, то хуже.

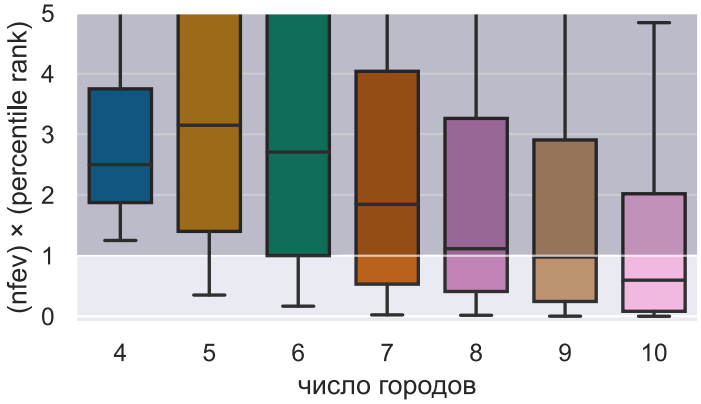


Рисунок 4.4 — Распределение величины $M \cdot p(c)$, где M — количество вычислений функции $\langle \text{cost} \rangle = f(\vec{\theta})$, $\vec{\theta}$ — вариационные параметры, $p(c)$ — процентильный ранг решения, к которому пришёл алгоритм. Незатенённая область соответствует случаю, когда $M \cdot p(c) < 1$ и разработанный алгоритм оказывается производительнее простого перебора.

Рисунок 4.4 показывает, что с ростом числа городов величина $M \cdot p(c)$ в среднем уменьшается, и при 10 городах она оказывается меньше единицы более чем в половине запусков.

Но нужно отметить, что при построении графика мы в качестве *nfev* брали количество вычислений функции $\langle \text{cost} \rangle$ — средней стоимости по суперпозиции.

Чтобы её вычислить, нужно измерить квантовое состояние несколько раз (в нашей симуляции *n_samples* = 100 раз).

Более честная версия графика 4.4 выглядела бы растянутой по оси Y в *n_samples* раз. Следовательно, при 10 городах квантовое преимущество всё ещё не будет достигнуто. Будет ли оно достигнуто при большем числе городов — открытый вопрос. Величина на графике 4.4 уменьшается, что позволяет надеяться, что она достигнет 1 даже при умножении на *n_samples*, однако мы не можем просимулировать цепи с таким большим количеством кубитов.

4.4 Используемые ресурсы

Нашему алгоритму требуется $\lceil \log_2 n! \rceil = O(n \log n)$ кубитов (где n — число городов), поскольку мы храним на квантовом компьютере одно целое число $x \in [0,\ n!)$.

Это значительно меньше, чем в других алгоритмах, которые, как правило, хранят маршрут в виде n^2 или даже n^3 бинарных переменных (см. подраздел 2.3 в «Обзоре литературы»):

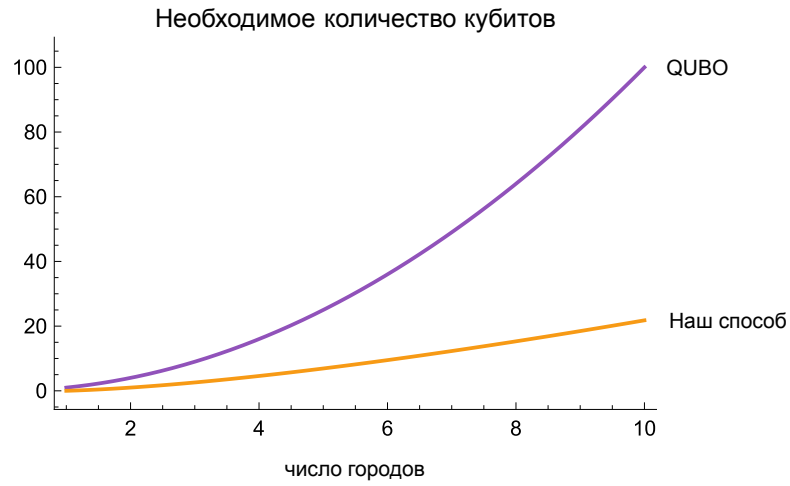


Рисунок 4.5 — Количество кубитов, необходимое для нашего алгоритма ($\log_2 n!$) и для алгоритмов, сводящих задачу коммивояжёра к задаче квадратичной бинарной оптимизации (n^2). Мы не стали дискретизировать данный график.

Осенью 2021 года компания IBM объявила о создании 127-кубитного квантового компьютера [24]. На таком компьютере наш алгоритм сможет искать решение задачи коммивояжёра с 33 городами, что соответствует $8.68 \cdot 10^{36}$ возможным маршрутам.

5 Проблемы алгоритма и направления будущих исследований

5.1 Получение решения, произвольно близкого к точному

Как видно из рисунка 4.2, наш алгоритм в случае случайных графов в среднем позволяет получать решения со стоимостью приблизительно вдвое меньше средней стоимости. Но что делать, если в практической задаче понадобится найти решение, более близкое к оптимальному?

Один из способов, которые теоретически могут сработать, — это использовать многослойную схему (рис. 5.1). Многослойная вариационная цепь подобна многослойной нейронной сети. У неё больше параметров, что может позволить ей лучше находить решения.

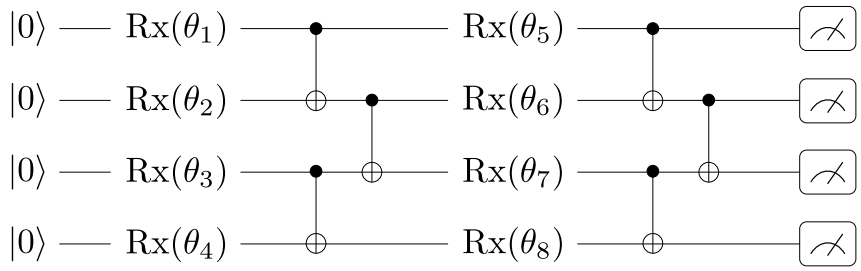


Рисунок 5.1 — Схема с двумя слоями. Количество слоёв можно увеличивать неограниченно. Заметим, что без гейтов CNOT увеличивать глубину цепи не получилось бы, так как два идущих подряд гейта Rx эквивалентны одному.

Мы протестировали двухслойную цепь на графе из 6 городов и сравнили её с однослойной (рис. 5.2):

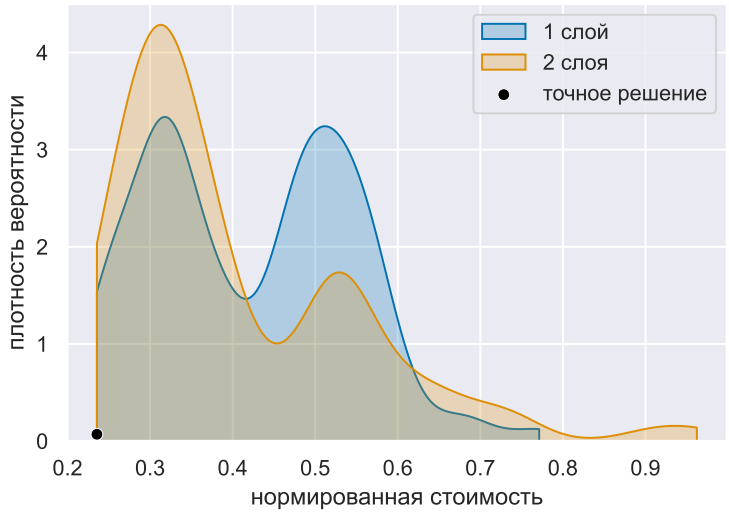


Рисунок 5.2 — Распределения стоимостей решений, которые получаются при использовании однослойной и двухслойной цепи.

Видно, что решения, полученные двухслойной цепью, чаще имеют меньшую стоимость, чем решения, полученные с помощью однослойной. Это имеет свою цену: в случае двухслойной цепи понадобилось больше повторных измерений (500 вместо 100), чтобы с достаточной точностью вычислять стоимость состояния. Дальнейшее увеличение числа слоёв к успеху не привело.

Возможно, имеет смысл использовать другие цепи и наборы параметрических гейтов. При этом можно опираться на работу [25] (2019), в которой есть обзор различных вариационных квантовых цепей без привязки к какой-либо конкретной задаче.

Кроме того, в будущих исследованиях имеет смысл рассмотреть различные критерии остановки (см. раздел 3.4). Возможно, именно правильный критерий остановки, учитывающий шумы, позволит неограниченно увеличивать число слоёв в цепи и точность алгоритма в целом.

5.2 Переиспользование подобранных параметров

В разделе «Обзор литературы» мы обсуждали QAOA — широко изученный вариационный квантовый алгоритм. Его сложно напрямую применить к задаче коммивояжёра, но у него есть одно преимущество по сравнению с разработанным нами алгоритмом.

Если решать схожие между собой задачи, отличающиеся только числами, то при использовании QAOA оптимальные вариационные параметры оказываются почти одинаковыми [26] (2020). Это позволяет использовать результат одной задачи в качестве начального приближения для другой, сильно уменьшая число итераций.

В нашем алгоритме это не так. Если мы возьмём подобранные параметры от другой задачи, то мы получим ответ для другой задачи.

Но попытка перенять это свойство QAOA может быть одним из направлений будущих исследований. Для этого потребуется:

1) отразить в квантовой цепи структуру задачи, например, в QAOA для этого применяют оператор $e^{-i\hat{C}t}$, где $\hat{C}$ — оператор стоимости.

2) использовать в качестве начального состояния равную суперпозицию.

На данный момент в нашем алгоритме в качестве начального используется состояние $|0\rangle^{\otimes m}$. Оно соответствует перестановке под номером 0.

Чтобы сгенерировать равную суперпозицию всех допустимых решений, нужно сгенерировать равную суперпозицию чисел от $|0\rangle$ до $|n! - 1\rangle$. Если бы $n!$ было степенью двойки, то можно было бы просто применить ко всем кубитам гейт Адамара. В общем же случае можно использовать обобщённый алгоритм усиления амплитуды, как показано в [27] (2019).

В той же работе [27] описан способ сгенерировать суперпозицию самих перестановок, а не только их номеров. Это может быть полезно для того, чтобы попытаться отразить в квантовой цепи структуру задачи.

5.3 Применение к другим задачам

Наш алгоритм легко обобщается на другие задачи, в которых решения-кандидаты могут быть каким-то образом пронумерованы. Точно так же квантовый компьютер будет использоваться для создания суперпозиции нескольких номеров, а на классическом компьютере будут вычисляться соответствующие стоимости.

Для этого достаточно, чтобы существовал эффективный классический алгоритм получения решения по его номеру. Как мы показали в этой работе, такие алгоритмы существуют, если допустимыми ответами на задачу являются перестановки. Помимо задачи коммивояжёра к таким задачам относится, например, комбинаторная задача о назначениях.

Другим классом подходящих задач являются те, в которых допустимыми ответами являются сочетания. Сочетания, как и перестановки, можно сортировать в лексикографическом порядке и затем восстанавливать решения по их номерам. В случае перестановок для этого используется факториальная, а в случае сочетаний — комбинаторная система счисления. Подробный анализ необходимых алгоритмов приведён в работе [28] (2021).

ЗАКЛЮЧЕНИЕ

Мы разработали вариационный квантовый алгоритм для задачи коммивояжёра. Мы оценили его сложность как $O(n^2 \log n)$, а количество требуемых кубитов как $O(n \log n)$.

Разработанный алгоритм является приближённым и не всегда позволяет получить точное решение. Это естественно, поскольку мы пытались решить неполиномиальную задачу за полиномиальное время. Однако, во-первых, по соотношению сложность / качество решений наш алгоритм превосходит простой перебор. Во-вторых, предварительные данные показывают, что точность алгоритма можно улучшить, как описано в разделе 5.1.

Наш алгоритм требует, чтобы кубиты на квантовом компьютере были соединены в линию. Такая простая топология позволяет использовать алгоритм на существующих квантовых компьютерах.

Также наша схема позволяет регулировать глубину цепи. В данной работе она поддерживалась низкой, чтобы адаптировать алгоритм к устройствам с небольшим временем когерентности. Мы показали, что даже неглубокие цепи, которые легко можно реализовать на современных квантовых компьютерах, позволяют добиться достаточной оптимальности решений.

В процессе создания алгоритма мы разработали новый способ представления маршрутов на квантовом компьютере. Он основан на том, что все возможные маршруты можно пронумеровать в лексикографическом порядке. Мы использовали тот факт, что на классическом компьютере по номеру перестановки можно быстро восстановить саму перестановку, переведя его в факториальную систему счисления. Наше представление автоматически учитывает нетривиальные ограничения задачи, касающиеся того, чтобы маршрут проходил через все города и не распадался на не связанные друг с другом петли.

Наш алгоритм можно обобщить на другие задачи, в которых решения можно каким-либо эффективным образом пронумеровать. К таким относится, например, комбинаторная задача о назначениях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Quantum computing at the quantum advantage threshold: a down-to-business review / A. K. Fedorov [et al.]. — 2022. — URL: https://arxiv.org/abs/2203.17181.

2 Variational quantum algorithms / M. Cerezo [et al.] // Nature Reviews Physics. — 2021. — URL: https://doi.org/10.1038/s42254-021-00348-9.

3 The Variational Quantum Eigensolver: a review of methods and best practices / J. Tilly [et al.]. — 2021. — URL: https://arxiv.org/abs/2111.05176.

4 Quantum approximate optimization algorithm for MaxCut: A fermionic view / Z. Wang [et al.] // Phys. Rev. A. — 2018. — URL: https://link.aps.org/doi/10.1103/PhysRevA.97.022304.

5 *Salehi Ö.*, *Glos A.*, *Miszczak J. A.* Unconstrained binary models of the travelling salesman problem variants for quantum optimization // Quantum Information Processing. — 2022. — URL: https://doi.org/10.1007/s11128-021-03405-5.

6 *González-Bermejo S.*, *Alonso-Linaje G.*, *Atchade-Adelomou P.* GPS: A new TSP formulation for its generalizations type QUBO. — 2021. — URL: https://arxiv.org/abs/2110.12158.

7 *Kuroiwa K.*, *Nakagawa Y. O.* Penalty methods for a variational quantum eigensolver // Phys. Rev. Research. — 2021. — URL: https://link.aps.org/doi/10.1103/PhysRevResearch.3.013197.

8 *Cirq Developers*. Cirq. — 2022. — URL: https://doi.org/10.5281/zenodo.6599601.

9 *Durr C.*, *Hoyer P.* A Quantum Algorithm for Finding the Minimum. — 1996. — URL: https://arxiv.org/abs/quant-ph/9607014.

10 Efficient quantum algorithm for solving travelling salesman problem: An IBM quantum experience / K. Srinivasan [et al.]. — 2018. — URL: https://arxiv.org/abs/1805.10928.

11 *Gilliam A.*, *Woerner S.*, *Gonciulea C.* Grover Adaptive Search for Constrained Polynomial Binary Optimization // Quantum. — 2021. — URL: https://doi.org/10.22331/q-2021-04-08-428.

12 Quantum Speedups for Exponential-Time Dynamic Programming Algorithms / A. Ambainis [et al.]. — 2018. — URL: https://arxiv.org/abs/1807.05209.

13 *Moylett A. E.*, *Linden N.*, *Montanaro A.* Quantum speedup of the traveling-salesman problem for bounded-degree graphs // Phys. Rev. A. — 2017. — URL: https://link.aps.org/doi/10.1103/PhysRevA.95.032323.

14 *Lucas A.* Ising formulations of many NP problems // Frontiers in Physics. — 2014. — URL: https://www.frontiersin.org/article/10.3389/fphy.2014.00005.

15 *Farhi E.*, *Goldstone J.*, *Gutmann S.* A Quantum Approximate Optimization Algorithm. — 2014. — URL: https://arxiv.org/abs/1411.4028.

16 *Radzihovsky M.*, *Murphy J.*, *Swofford M.* A QAOA solution to the traveling salesman problem using pyQuil. — 2019. — URL: https://cs269q.stanford.edu/projects2019/radzihovsky_murphy_swofford_Y.pdf.

17 *Henry D.*, *Dudas S. J. L.* Solving the Traveling Salesman Problem Using QAOA. — 2019. — URL: https://cs269q.stanford.edu/projects2019/DudasHenry_Y.pdf.

18 *Ceroni J.* Fun With Graphs and QAOA. — 2020. — URL: https://lucaman99.github.io/new_blog/2020/mar16.html.

19 From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz / S. Hadfield [et al.] // Algorithms. — 2019. — URL: https://doi.org/10.3390/a12020034.

20 The Quantum Approximate Algorithm for Solving Traveling Salesman Problem / Y. Ruan [et al.] // Computers, Materials & Continua. — 2020. — URL: https://doi.org/10.32604/cmc.2020.010001.

21 *Ostaszewski M.*, *Grant E.*, *Benedetti M.* Structure optimization for parameterized quantum circuits // Quantum. — 2021. — URL: https://doi.org/10.22331/q-2021-01-28-391.

22 TensorFlow Quantum: A Software Framework for Quantum Machine Learning / M. Broughton [et al.]. — 2020. — URL: https://arxiv.org/abs/2003.02989.

23 *Rubinstein R. Y.*, *Kroese D. P.* // The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine

Learning. — Springer New York, 2004. — P. 203–225. — ISBN 978-1-4757-4321-0. — URL: https://doi.org/10.1007/978-1-4757-4321-0_6.

24 *Dial O.* Eagle's quantum performance progress // IBM Research Blog. — 2022. — URL: https://research.ibm.com/blog/eagle-quantum-processor-performance.

25 *Sim S.*, *Johnson P. D.*, *Aspuru-Guzik A.* Expressibility and Entangling Capability of Parameterized Quantum Circuits for Hybrid Quantum-Classical Algorithms // Advanced Quantum Technologies. — 2019. — URL: https://doi.org/10.1002/qute.201900070.

26 Quantum Approximate Optimization Algorithm: Performance, Mechanism, and Implementation on Near-Term Devices / L. Zhou [et al.] // Phys. Rev. X. — 2020. — June. — URL: https://link.aps.org/doi/10.1103/PhysRevX.10.021067.

27 Graph comparison via nonlinear quantum search / M. Chiew [et al.] // Quantum Information Processing. — 2019. — URL: https://doi.org/10.1007/s11128-019-2407-2.

28 *Genitrini A.*, *Pépin M.* Lexicographic Unranking of Combinations Revisited // Algorithms. — 2021. — DOI: 10.3390/a14030097. — URL: https://www.mdpi.com/1999-4893/14/3/97.