

Министерство науки и высшего образования РФ  
ФГБОУ ВО «Тверской государственный университет»  
Математический факультет  
Направление 02.04.01 Математика и компьютерные науки  
Профиль «Математическое и компьютерное моделирование»

## МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Вариационный квантовый алгоритм с оптимизацией  
методом отжига

Автор:  
Алешин Д.А.  
Подпись:

Научный руководитель:  
д. ф.-м. н. Цирулёв А.Н.  
Подпись:

Допущен к защите:  
Руководитель ООП: Цветков В.П.

---

*(подпись, дата)*

Тверь 2025

# Оглавление

Введение	4
<b>1 Общая схема квантовых вариационных алгоритмов</b>	<b>5</b>
1.1 Базис Паули . . . . .	5
1.1.1 Связь стандартного базиса и базиса Паули . . . . .	5
1.1.2 Коммутационное и антикоммутационное соотношение	8
1.1.3 Коммутирующие элементы в алгебре Ли группы $SU(2^n)$ . . . . .	9
1.2 Вариационная квантовая оптимизация . . . . .	11
1.3 Общая схема алгоритма и анзац . . . . .	14
1.4 Пример, иллюстрирующий особенности алгоритма . . . . .	14
1.5 Оптимизация . . . . .	19
<b>2 Вариационный квантовый алгоритм на основе метода отжига</b>	<b>20</b>
2.1 Метод отжига . . . . .	20
2.2 Алгоритм . . . . .	22
2.2.1 Этап 1: Чтение и парсинг данных гамильтониана .	22
2.2.2 Этап 2: Инициализация параметров . . . . .	23
2.2.3 Этап 3: Вычисление энергии . . . . .	24
2.2.4 Этап 4: Алгоритм отжига . . . . .	25
2.3 Сравнительные результаты тестирования . . . . .	27
<b>Заключение</b>	<b>27</b>

Литература	28
Приложения	31

# Введение

# Глава 1

## Общая схема квантовых вариационных алгоритмов

### 1.1 Базис Паули

#### 1.1.1 Связь стандартного базиса и базиса Паули

Рассмотрим квантовую систему из  $n$  кубитов, где каждый кубит связан с двумерным гильбертовым пространством  $\mathcal{H}$  и его эрмитово сопряжённым пространством  $\mathcal{H}^\dagger$ . Обозначим через  $\mathcal{H}_n = \mathcal{H}^{\otimes n}$  и  $\mathcal{H}_n^\dagger = (\mathcal{H}^\dagger)^{\otimes n}$  гильбертово пространство системы и его эрмитово сопряжение соответственно. Пространство линейных операторов, действующих на  $\mathcal{H}$  и  $\mathcal{H}^\dagger$  левым и правым умножением, задаётся как  $L(\mathcal{H}_n) = \mathcal{H}_n \otimes \mathcal{H}_n^\dagger$ . Тогда

$$\dim_{\mathbb{C}} \mathcal{H}_n = \dim_{\mathbb{C}} \mathcal{H}_n^\dagger = 2^n, \quad \dim_{\mathbb{C}} L(\mathcal{H}_n) = 2^{2n}.$$

Пространство  $L(\mathcal{H}_n)$  наделено скалярным произведением Гильберта-Шмидта:

$$\langle \hat{A}, \hat{B} \rangle = \text{tr}(\hat{A}^\dagger \hat{B}), \quad \hat{A}, \hat{B} \in L(\mathcal{H}_n), \quad (1.1)$$

которое естественно продолжает скалярное произведение в  $\mathcal{H}_n$ . вещественное линейное пространство эрмитовых операторов далее обозначим как  $H(\mathcal{H}_n)$ .

Пусть  $\{|0\rangle, |1\rangle\}$  образуют ортонормированный базис в однокубитном пространстве  $\mathcal{H}$ . Единичная матрица и матрицы Паули задаются как:

$$\sigma_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \sigma_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \sigma_2 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix},$$

а соответствующие операторы Паули представляются в виде:

$$\begin{aligned} \hat{\sigma}_0 &= |0\rangle\langle 0| + |1\rangle\langle 1|, \quad \hat{\sigma}_1 = |0\rangle\langle 1| + |1\rangle\langle 0|, \\ \hat{\sigma}_2 &= -i|0\rangle\langle 1| + i|1\rangle\langle 0|, \quad \hat{\sigma}_3 = |0\rangle\langle 0| - |1\rangle\langle 1|. \end{aligned}$$

Эти операторы одновременно эрмитовы и унитарны, а также образуют базис в  $\mathcal{H}$ . Обратное преобразование выражается следующим образом:

$$|0\rangle\langle 0| = \frac{\hat{\sigma}_0 + \hat{\sigma}_3}{2}, \quad |0\rangle\langle 1| = \frac{\hat{\sigma}_1 + i\hat{\sigma}_2}{2}, \quad |1\rangle\langle 0| = \frac{\hat{\sigma}_1 - i\hat{\sigma}_2}{2}, \quad |1\rangle\langle 1| = \frac{\hat{\sigma}_0 - \hat{\sigma}_3}{2}.$$

Для  $k, l, m \in \{1, 2, 3\}$  выполняются свойства:  $\text{tr} \hat{\sigma}_k = 0$ ,  $\hat{\sigma}_k^2 = \hat{\sigma}_0$ , а также

$$\hat{\sigma}_k \hat{\sigma}_l = -\hat{\sigma}_l \hat{\sigma}_k, \quad \hat{\sigma}_k \hat{\sigma}_l = i \text{sign}(\pi) \hat{\sigma}_m, \quad (klm) = \pi(123), \quad (1.2)$$

где  $\pi(123)$  — произвольная перестановка множества  $\{1, 2, 3\}$ .

Рассмотрим стандартный<sup>1</sup> бинарный базис в  $\mathcal{H}_n$ , образованный ортонормированными базисами  $\{|0\rangle, |1\rangle\}$  в однокубитных пространствах. Позиция в тензорном произведении позволяет различать кубиты. Для фиксированного  $n$  элементы этого базиса и соответствующие им элементы двумерного базиса удобно записывать как:

$$|k\rangle = |k_1 \dots k_n\rangle = |k_1\rangle \otimes \dots \otimes |k_n\rangle, \quad \langle k| = \langle k_1 \dots k_n| = \langle k_1| \otimes \dots \otimes \langle k_n|,$$

где строки  $k_1 \dots k_n$  ( $k_1, \dots, k_n \in \{0, 1\}$ ) интерпретируются как двоичные числа с десятичным представлением  $k$ . Например,  $|101\rangle = |5\rangle$  и  $|00110\rangle = |6\rangle$ .

---

<sup>1</sup>Мы избегаем термина «вычислительный», так как он может приводить к неоднозначности. И базис Паули, и стандартный базис являются вычислительными в одинаковом контексте.

В стандартном базисе:

$$|u\rangle = \sum_{k=0}^{2^n-1} u_k |k\rangle, \quad \hat{A} = \sum_{k,l=0}^{2^n-1} a_{kl} |k\rangle\langle l|,$$

где  $|u\rangle \in \mathcal{H}_n$  и  $\hat{A} \in L(\mathcal{H}_n)$ .

Базис Паули  $P(\mathcal{H}_n)$  в  $L(\mathcal{H}_n)$  определяется как:

$$\{\hat{\sigma}_{k_1 \dots k_n}\}_{k_1, \dots, k_n \in \{0,1,2,3\}}, \quad \hat{\sigma}_{k_1 \dots k_n} = \hat{\sigma}_{k_1} \otimes \dots \otimes \hat{\sigma}_{k_n}, \quad (1.3)$$

где  $\hat{\sigma}_{0\dots 0}$  — тождественный оператор. Базис  $P(\mathcal{H}_n)$  содержит  $4^n$  элементов. Для краткости будем использовать обозначение:

$$\hat{\sigma}_K = \hat{\sigma}_{k_1 \dots k_n},$$

где строка Паули  $k_1 \dots k_n$  ( $k_1, \dots, k_n \in \{0, 1, 2, 3\}$ ) соответствует числу  $K$  в десятичной системе ( $0 \leq K \leq 4^n - 1$ ). Строка Паули  $K$  и элемент  $\hat{\sigma}_K$  взаимно однозначно соответствуют друг другу.

Сравним  $P(\mathcal{H}_n)$  со стандартным базисом. Для элементов базиса Паули выполняются:

$$\hat{\sigma}_{k_1 \dots k_n} \hat{\sigma}_{k_1 \dots k_n} = \hat{\sigma}_{0\dots 0}, \quad \text{tr } \hat{\sigma}_{0\dots 0} = 2^n, \quad \text{tr } \hat{\sigma}_{k_1 \dots k_n} \Big|_{k_1 \dots k_n \neq 0\dots 0} = 0. \quad (1.4)$$

Базис Паули является эрмитовым, унитарным и ортогональным относительно скалярного произведения (1.1). Отметим, что оператор  $|k\rangle\langle l|$  из стандартного базиса не является унитарным или эрмитовым при  $k \neq l$ . Стандартный базис не включает тождественный оператор, который в этом базисе записывается как:

$$\sum_{k=0}^{2^n-1} |k\rangle\langle k|.$$

В базисе Паули любой оператор  $\hat{U}$  из унитарной группы  $U(\mathcal{H}_n)$  (где  $\hat{U}^\dagger \hat{U} = \hat{\sigma}_{0\dots 0}$ ) раскладывается в виде:

$$\hat{U} = \sum_{i_1, \dots, i_n \in \{0,1,2,3\}} U_{i_1 \dots i_n} \hat{\sigma}_{i_1 \dots i_n}, \quad \hat{U}^\dagger = \sum_{i_1, \dots, i_n \in \{0,1,2,3\}} \overline{U}_{i_1 \dots i_n} \hat{\sigma}_{i_1 \dots i_n},$$

где коэффициенты удовлетворяют условиям:

$$\sum_{i_1, \dots, i_n \in \{0,1,2,3\}} \bar{U}_{i_1 \dots i_n} U_{i_1 \dots i_n} = 1, \quad \sum_{\substack{i_1, \dots, i_n, j_1, \dots, j_n \in \{0,1,2,3\} \\ (i_1, \dots, i_n) \neq (j_1, \dots, j_n)}} \bar{U}_{i_1 \dots i_n} U_{j_1 \dots j_n} = 0.$$

Последнее условие эквивалентно  $2^{2n-1}(2^n - 1)$  независимым соотношениям.

Эрмитовы операторы в базисе Паули разлагаются с вещественными коэффициентами.

### 1.1.2 Коммутационное и антикоммутационное соотношение

Коммутатор определяет взаимодействие операторов при их перестановке. Для операторов  $A$  и  $B$  он задаётся как:

$$[A, B] = AB - BA.$$

Если  $[A, B] = 0$ , операторы коммутируют; в противном случае — нет. В базисе Паули коммутаторы выражаются через символ Леви-Чивиты  $\varepsilon_{ijk}$ :

$$[\sigma_i, \sigma_j] = 2i\varepsilon_{ijk}\sigma_k,$$

где  $\varepsilon_{ijk}$  равен 1 при чётной перестановке индексов  $(i, j, k)$ ,  $-1$  при нечётной и 0 в остальных случаях. Например:

$$[\sigma_1, \sigma_2] = 2i\sigma_3, \quad [\sigma_2, \sigma_3] = 2i\sigma_1.$$

Антикоммутатор характеризует симметричное произведение операторов:

$$\{A, B\} = AB + BA.$$

Если  $\{A, B\} = 0$ , операторы антикоммутируют. Для операторов Паули:

$$\{\sigma_i, \sigma_j\} = 2\delta_{ij},$$

где  $\delta_{ij}$  — символ Кронекера (1 при  $i = j$ , 0 иначе). Примеры:

$$\{\sigma_1, \sigma_2\} = 0, \quad \{\sigma_2, \sigma_3\} = 0.$$



Коммутаторы и антикоммутаторы применяются в квантовой механике для анализа свойств систем (спин электрона, кубиты), в квантовой теории поля (взаимодействия частиц) и квантовых вычислениях (алгоритмы, коррекция ошибок). Коммутаторы помогают определить совместную измеримость наблюдаемых, а антикоммутаторы — описать фермионные системы.

### 1.1.3 Коммутирующие элементы в алгебре Ли группы $SU(2^n)$

Рассмотрим базис Паули и множество строк Паули длины  $n$ :

$$\text{Str}_n = \{K = k_1 \dots k_n\}_{k_1, \dots, k_n \in \{0, 1, 2, 3\}}.$$

1. Множество  $\mathbb{F}_4 = \{0, 1, 2, 3\}$  образует квадгруппу Клейна с умножением:

$$0 \cdot k = k, \quad k \cdot k = 0, \quad k \cdot l = m,$$

где  $k, l, m \in \{1, 2, 3\}$  и  $(klm)$  — произвольная перестановка  $(1\ 2\ 3)$ .

2. Функция  $s : \mathbb{F}_4 \times \mathbb{F}_4 \rightarrow \{1, i, -i\}$  задаётся значениями:

$$\begin{aligned} s(0, 0) = s(0, k) = s(k, 0) = s(k, k) &= 1, \quad k = 1, 2, 3, \\ s(1, 2) = s(2, 3) = s(3, 1) &= i, \quad s(2, 1) = s(3, 2) = s(1, 3) = -i. \end{aligned}$$

3. Функция  $S : \text{Str}_n \times \text{Str}_n \rightarrow \{1, -1, i, -i\}$  определяется как:

$$S_{KL} = s(k_1, l_1) \cdot s(k_2, l_2) \cdot \dots \cdot s(k_n, l_n),$$

где  $K = k_1 k_2 \dots k_n$  и  $L = l_1 l_2 \dots l_n$ .

Симметрия функции  $S$  зависит от числа пар  $(k_r, l_r)$  (на позициях  $r$  в строках  $K$  и  $L$ ), где  $k_r, l_r \in \{1, 2, 3\}$  и  $k_r \neq l_r$ , а также от их взаимного порядка. Пусть  $\omega_{KL}^+$  и  $\omega_{KL}^-$  — количество пар вида  $(1, 2), (2, 3), (3, 1)$  и  $(2, 1), (3, 2), (1, 3)$  соответственно, и  $\omega_{KL} = \omega_{KL}^+ + \omega_{KL}^-$ . Тогда

$$S_{(KL)} = \frac{S_{KL}}{2} (1 + (-1)^{\omega_{KL}}), \quad S_{[KL]} = \frac{S_{KL}}{2} (1 - (-1)^{\omega_{KL}}), \quad (1.5)$$

где

$$S_{KL} = i^{\omega_{KL}} (-1)^{\omega_{KL}^-}.$$

Здесь  $S_{(KL)}$  и  $S_{[KL]}$  — симметричная и антисимметричная части  $S_{KL}$ . Значения  $S_{KL}$ ,  $S_{(KL)}$  и  $S_{[KL]}$  приведены в таблице 2.

$\omega_{KL} \bmod 4$	0	2	0	2	1	3	1	3
$\omega_{KL}^- \bmod 4$	0	1	1	0	0	1	1	0
$S_{KL}$		1	-1	-1	$i$	$i$	$-i$	$-i$
$S_{(KL)}$	1	1	-1	-1	0	0	0	0
$S_{[KL]}$	0	0	0	0	$i$	$i$	$-i$	$-i$

Таблица 1: Множитель до  $\hat{\sigma}_M$  в (1.6) для  $\hat{\sigma}_K \hat{\sigma}_L$ ,  $\{\hat{\sigma}_K, \hat{\sigma}_L\}$ , и  $[i\hat{\sigma}_K, i\hat{\sigma}_L]$ .

Композицию элементов базиса Паули, их антикоммутаторов и коммутаторов можно компактно выразить в виде, удобном для программной реализации:

$$\hat{\sigma}_K \hat{\sigma}_L = S_{KL} \hat{\sigma}_M, \quad \{\hat{\sigma}_K, \hat{\sigma}_L\} = S_{(KL)} \hat{\sigma}_M, \quad [i\hat{\sigma}_K, i\hat{\sigma}_L] = -S_{[KL]} \hat{\sigma}_M, \quad (1.6)$$

где

$$\hat{\sigma}_M = \hat{\sigma}_{m_1 \dots m_n}, \quad m_r = k_r \cdot l_r \quad (r = 1, \dots, n). \quad (1.7)$$

Две строки Паули длины  $n$  могут коммутировать, даже имея ненулевые элементы в одних и тех же позициях. Например, операторы  $\hat{\sigma}_{11}$ ,  $\hat{\sigma}_{22}$  и  $\hat{\sigma}_{33}$  коммутируют друг с другом. Унитарная матрица перехода из стандартного базиса  $\{|i_1 \dots i_n\rangle \langle j_1 \dots j_n|\}$  в базис Паули содержит только элементы 0,  $\pm 1$  и  $\pm i$ . Например:

$$|00 \dots 0\rangle \langle 00 \dots 0| \rightarrow \frac{1}{2^n} \sum_{i_1, \dots, i_n \in \{0,3\}} \hat{\sigma}_{i_1 \dots i_n}.$$

Общее выражение для стандартных ортогональных проекторов имеет вид:

$$|i_1 \dots i_n\rangle \langle i_1 \dots i_n| = \frac{1}{2^n} \sum_{k_1, \dots, k_n \in \{0,3\}} \mathcal{X}_{k_1}^{i_1} \dots \mathcal{X}_{k_n}^{i_n} \hat{\sigma}_{k_1 \dots k_n},$$

где

$$\mathcal{X}_0^0 = \mathcal{X}_3^0 = \mathcal{X}_0^1 = 1, \quad \mathcal{X}_3^1 = -1.$$

Из выражения (1.6) следует, что: 1. Множество  $\{i\hat{\sigma}_K\}_{K=0}^{4^n-1}$  образует ортонормированный базис в  $\mathfrak{su}(2^n)$ . 2. Множество

$$\tilde{P}(\mathcal{H}_n) = \{\epsilon\hat{\sigma}_K \mid K \in \text{Str}_n, \epsilon \in \{\pm 1, \pm i\}\},$$

содержащее  $4^{n+1}$  элементов, образует группу — т.н.  $n$ -кубитную группу Паули.

Нормализатор группы Паули в унитарной группе:

$$\mathcal{C}(\mathcal{H}_n) = \left\{ \hat{U} \in U(\mathcal{H}_n) \mid \hat{U}\hat{\sigma}_K\hat{U}^\dagger \in \tilde{P}(\mathcal{H}_n), \forall \hat{\sigma}_K \in \tilde{P}(\mathcal{H}_n) \right\},$$

называется группой Клиффорда. Исходя из (1.2), (1.4) и (1.7) получаем следующее утверждение

**Утверждение 1.** *Взаимные унитарные преобразования базисных операторов Паули подчиняются соотношениям  $\hat{\sigma}_{i_1\dots i_n}\hat{\sigma}_{k_1\dots k_n}\hat{\sigma}_{i_1\dots i_n} = \pm\hat{\sigma}_{i_1\dots i_n}$ , где знак плюс стоит тогда и только тогда, когда количество троек  $(i_mk_mi_m)_{m \in \{1,\dots,n\}}$ , удовлетворяют условиям  $i_m \neq k_m$ ,  $i_m \neq 0$ , и  $k_m \neq 0$  четности.*

$l$	0	1	2	3	4	5	6	7
$l_2l_1l_0$	000	001	010	011	100	101	110	111
$k_2k_1k_0$	011	011	011	011	011	011	011	011
$\bar{l} \wedge k$	011	010	001	000	011	010	001	000
$l \wedge k$	000	001	010	011	000	001	010	011
$l \wedge \bar{k}$	000	000	000	000	100	100	100	100
$\hat{\sigma}_I$	$\hat{\sigma}_{011}$	$\hat{\sigma}_{012}$	$\hat{\sigma}_{021}$	$\hat{\sigma}_{022}$	$\hat{\sigma}_{311}$	$\hat{\sigma}_{312}$	$\hat{\sigma}_{121}$	$\hat{\sigma}_{322}$

Таблица 2: Элементы базиса Паули, возникающие для  $k = 011$ .

## 1.2 Вариационная квантовая оптимизация

Пусть  $\mathcal{H}_n$  — гильбертово пространство квантовой системы, состоящей из  $n$  кубитов,  $\mathcal{S} \subset \mathcal{H}_n$  — пространство векторов состояния (т.е. векторов, нормированных на единицу),  $L(\mathcal{H}_n)$  — алгебра операторов на  $\mathcal{H}_n$  и  $\hat{H} \in L(\mathcal{H}_n)$  — эрмитов оператор. Определим функцию

$$E(|u\rangle) = \langle u | \hat{H} | u \rangle, \quad |u\rangle \in \mathcal{S}. \quad (1.8)$$

В простейшей постановке **квантовой задачи оптимизации** требуется найти вектор состояния, на котором целевая функция (функция стоимости)  $E$  принимает минимальное значение, т.е., в формальной записи, решить задачу

$$E(|u\rangle) \xrightarrow{|u\rangle \in \mathcal{S}} \min. \quad (1.9)$$

Ниже, для определенности и краткости, будем называть  $\hat{H}$  гамильтонианом системы, а целевую функцию  $E$  — энергией.

Сложность алгоритмов прямого вычисления собственных значений гамильтониана  $\hat{H}$  растет экспоненциально с ростом числа кубитов, поэтому для больших систем используются вариационные методы решения задачи оптимизации (1.9).

Вариационными квантовыми алгоритмами обычно называют такие гибридные квантово-классические алгоритмы, нацеленные на решение квантовых задач оптимизации посредством квантовых вычислений или их классической имитации, которые проводят вариационную настройку параметров квантовой схемы. Параметрически управляемое квантовое устройство, обычно представленное квантовой цепью, реализует анзац, т.е. унитарное преобразование стандартного начального состояния  $|0\rangle^{\otimes n}$  или, как вариант, предудущего полученного состояния. На каждом шаге регулирующие параметры подбираются так, чтобы минимизировать энергию (целевую функцию). Обычно это выполняется путём измерения энергии состояний, предоставляемых вариационной схемой, и обновления параметров для минимизации целевой функции.

В точной математической формулировке сказанное означает, что в функции (1.8) вектор состояния  $|u\rangle$  зависит от набора  $m$  параметров  $\boldsymbol{\theta} = (\theta_1, \dots, \theta_m)$ , которые принимают значения в некоторой связной и односвязной области  $\Omega \in \mathbb{R}^m$ . Вариационная формулировка квантовой задачи оптимизации (1.9) имеет вид

$$E(|u(\boldsymbol{\theta})\rangle) \xrightarrow{\boldsymbol{\theta} \in \Omega} \min, \quad E(|u(\boldsymbol{\theta})\rangle) = \langle u(\boldsymbol{\theta}) | \hat{H} | u(\boldsymbol{\theta}) \rangle. \quad (1.10)$$

Итак, цель вариационного квантового алгоритма — найти такой набор параметров, на котором энергия достигает минимума. Число параметров  $m$  в наборе  $\boldsymbol{\theta} = (\theta_1, \dots, \theta_m)$  зависит от конкретной задачи, в частности,

от числа кубитов в квантовом устройстве. Для  $n$  кубитов размерность пространства состояний,  $N = 2^n$ , растет экспоненциально с ростом числа кубитов. Поэтому вариационный квантовый алгоритм должен быть организован и выполнен так, чтобы выполнялось условие  $m \ll N$ , поскольку в противном случае высокий класс сложности алгоритма сделает его неэффективным с практической точки зрения.

Но наиболее важным вопросом является выбор зависимости вектора состояния  $|u(\boldsymbol{\theta})\rangle$  от параметров. В вариационных квантовых алгоритмах используется *анзац* (унитарное преобразование) вида

$$|u(\boldsymbol{\theta})\rangle = \hat{U}(\boldsymbol{\theta}) |u_0\rangle. \quad (1.11)$$

В общем случае форма анзаца определяет, какими будут параметры  $\boldsymbol{\theta}$  и как их можно настроить для минимизации энергии (целевой функции). Структура анзаца, как правило, будет зависеть от поставленной задачи, так как во многих случаях можно использовать информацию о проблеме, чтобы подобрать анзац: это "анзац, подсказанный задачей". Однако можно построить анзацы достаточно общего вида, которые пригодны для использования в некоторых классах задач даже тогда, когда интуиция и известная информация о задаче не позволяют его уточнить. Стандартно анзац выбирается в виде композиции  $m$  последовательно примененных унитарных преобразований

$$\hat{U}(\boldsymbol{\theta}) = \hat{U}_m(\theta_m) \cdots \hat{U}_1(\theta_1). \quad (1.12)$$

В композиции (1.12) выбор операторов определяется типом задачи и технической возможностью их реализации на конкретном квантовом устройстве. Например, можно выбрать

$$\hat{U}_K(\theta_K) = \hat{W}_K \exp(i\theta_K \hat{\sigma}_K) = \hat{W}_K (\cos \theta_K \hat{\sigma}_{0\dots 0} + i \sin \theta_K \hat{\sigma}_K), \quad (1.13)$$

где  $1 \leq K \leq m$ ,  $\hat{\sigma}_K = \hat{\sigma}_{k_1} \otimes \dots \otimes \hat{\sigma}_{k_n}$ ,  $k_1, \dots, k_n \in \{0, 1, 2, 3\}$ ,  $K = k_1 \dots k_n$  (т.е.  $K$  — десятичное представление строки  $k_1 \dots k_n$ , рассматриваемой как число по основанию 4),  $n$  — число кубитов, а  $\hat{W}_K$  — независимый от параметров унитарный оператор. Как правило, в строке  $k_1 \dots k_n$  только отдельные числа отличны от нуля, так что в тензорном произведении  $\hat{\sigma}_{k_1} \otimes \dots \otimes \hat{\sigma}_{k_n}$  часть операторов являются тождественными.

В другом распространенном варианте операторы в композиции (1.12) имеют вид

$$\hat{U}_K(\theta_K) = \hat{W}_K(e^{i\theta_{k_1}\hat{\sigma}_{k_1}} \otimes \dots \otimes e^{i\theta_{k_n}\hat{\sigma}_{k_n}}), \quad (1.14)$$

где по-прежнему  $1 \leq K \leq m$  и  $K = k_1 \dots k_n$ . Если в (1.13) все операторы  $\hat{W}_K$  могут быть тождественными, то в (1.14), по крайней мере некоторые операторы  $\hat{W}_K$  должны быть запутывающими и, следовательно, как минимум двухкубитными.

Таким образом, анзацы (1.12), (1.13) и (1.14) конкретизируют вариационную квантовую задачу оптимизации (1.10) и (1.11) в отношении параметрической зависимости вектора состояния,

$$|u(\boldsymbol{\theta})\rangle = \hat{U}(\boldsymbol{\theta})|0 \dots 0\rangle,$$

где начальное состояние имеет вид  $|0 \dots 0\rangle = |0\rangle^{\otimes n}$ . Если предположить далее, что мы в состоянии уверенно приготовить начальное состояние, реализовать анзац на физическом устройстве и вычислить значение энергии  $E(|u(\boldsymbol{\theta})\rangle) = \langle u(\boldsymbol{\theta}) | \hat{H} | u(\boldsymbol{\theta}) \rangle$  посредством измерений (с привлечением классического компьютера), то следующий — основной — вопрос можно сформулировать так: как искать параметры, которые обеспечивают глобальный минимум энергии. Этот этап выполняется с помощью классического компьютера, так что вариационный квантовый алгоритм — гибридный квантово-классический алгоритм: параметризованная квантовая схема и измерительный прибор представляют квантовую часть, а алгоритм настройки параметров — классическую.

## 1.3 Общая схема алгоритма и анзац

## 1.4 Пример, иллюстрирующий особенности алгоритма

Для иллюстрации алгоритма рассмотрим гамильтониан

$$\hat{H} = 2\hat{\sigma}_{03} + \hat{\sigma}_{30} - 4\hat{\sigma}_{11}, \quad (1.15)$$

который в стандартном базисе  $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$  имеет матрицу

$$H = \begin{pmatrix} 3 & 0 & 0 & -4 \\ 0 & -1 & -4 & 0 \\ 0 & -4 & 1 & 0 \\ -4 & 0 & 0 & -3 \end{pmatrix}.$$

Используя систему Maple находим собственные значения и собственные состояния в порядке возрастания собственных значений, начиная с основного состояния  $|u_0\rangle$  с собственным значением  $E_0$ :

$$E_0 = -5, \quad |u_0\rangle = \frac{1}{\sqrt{5}} |00\rangle + \frac{2}{\sqrt{5}} |11\rangle, \quad (1.16)$$

$$E_1 = -\sqrt{17}, \quad |u_1\rangle = \frac{\sqrt{17} + 1}{\sqrt{34 + 2\sqrt{17}}} |01\rangle + \frac{\sqrt{8}}{\sqrt{17 + \sqrt{17}}} |10\rangle, \quad (1.17)$$

$$E_2 = \sqrt{17}, \quad |u_1\rangle = -\frac{\sqrt{17} - 1}{\sqrt{34 - 2\sqrt{17}}} |01\rangle + \frac{\sqrt{8}}{\sqrt{17 - \sqrt{17}}} |10\rangle, \quad (1.18)$$

$$E_3 = 5, \quad |u_3\rangle = -\frac{2}{\sqrt{5}} |00\rangle + \frac{1}{\sqrt{5}} |11\rangle. \quad (1.19)$$

Рассмотрим далее пошаговое выполнение вариационного квантового алгоритма, который позволяет найти состояние, близкое к основному.

*Первый шаг — выбор анзаца*, т.е. унитарного преобразования  $\hat{U}(\boldsymbol{\theta})$ . В гамильтониан (1.15) не входят операторы вида  $\hat{\sigma}_{k2}$  и  $\hat{\sigma}_{2k}$  с  $k \neq 2$ , поэтому имеет смысл сразу выбирать анзац так, чтобы при действии на  $k \neq 2$  он давал вектор состояния с вещественными коэффициентами. Других наводящих соображений относительно формы анзаца не видно, поэтому следует рассмотреть разные варианты. В общем случае вектор параметров  $\boldsymbol{\theta}$  четырехмерен. В простейшем варианте анзац с четырехмерным вектором параметров  $\boldsymbol{\theta} = (\xi, \lambda, \mu, \nu)$  можно выбирать как композицию экспонент

$$\hat{U}(\boldsymbol{\theta}) = e^{i\xi\hat{\sigma}_{02}} e^{i\lambda\hat{\sigma}_{03}} e^{i\mu\hat{\sigma}_{30}} e^{i\nu\hat{\sigma}_{11}} \quad (1.20)$$

операторов Паули, присутствующих в гамильтониане (1.15). Вычислим

вначале

$$\begin{aligned}
e^{i\mu\hat{\sigma}_{30}}e^{i\nu\hat{\sigma}_{11}}|00\rangle &= (\cos\mu\hat{\sigma}_{00} + i\sin\mu\hat{\sigma}_{30})(\cos\nu|00\rangle + i\sin\nu|11\rangle) \\
&= \cos\mu\cos\nu|00\rangle - \sin\mu\sin\nu|11\rangle + i\sin\mu\cos\nu|00\rangle + i\cos\mu\sin\nu|11\rangle \\
&= e^{i\mu}\cos\nu|00\rangle + ie^{i\mu}\sin\nu|11\rangle = e^{i\mu}\cos\nu|00\rangle + e^{i(\mu+\pi/2)}\sin\nu|11\rangle.
\end{aligned}$$

Действуя на результат оператором  $e^{i\lambda\hat{\sigma}_{03}}$ , получим следующий промежуточный вектор состояния:

$$\begin{aligned}
e^{i\lambda\hat{\sigma}_{03}}e^{i\mu\hat{\sigma}_{30}}e^{i\nu\hat{\sigma}_{11}}|00\rangle &= e^{i\mu}\cos\nu(\cos\lambda\hat{\sigma}_{00} + i\sin\lambda\hat{\sigma}_{03})|00\rangle \\
&\quad + e^{i(\mu+\pi/2)}\sin\nu(\cos\lambda\hat{\sigma}_{00} + i\sin\lambda\hat{\sigma}_{03})|11\rangle \\
&= e^{i\mu}\cos\nu(\cos\lambda + i\sin\lambda)|00\rangle + e^{i(\mu+\pi/2)}\sin\nu(\cos\lambda + i\sin\lambda)|11\rangle \\
&= e^{i(\mu+\lambda)}\cos\nu|00\rangle + e^{i(\mu+\pi/2-\lambda)}\sin\nu|11\rangle. \quad (1.21)
\end{aligned}$$

Очевидно, что этот анзац не является универсальным.

Варьируя параметры  $\lambda, \mu, \nu$ , можно получить основное состояние (1.16) с точностью до несущественного множителя  $e^{i(\mu+\pi/4)}$ , например, при

$$\lambda = \pi/4, \quad \mu \in \mathbb{R}, \quad \cos\nu = 1/\sqrt{5}, \quad \sin\nu = 2/\sqrt{5}. \quad (1.22)$$

Здесь  $\mu$  — любое, поэтому к нужному результату приводит более простой анзац (при  $\mu = 0$ )  $\hat{U}(\boldsymbol{\theta}) = e^{i\lambda\hat{\sigma}_{03}}e^{i\nu\hat{\sigma}_{11}}$ , однако заранее это нам не известно. Более того основное состояние (1.16) можно достигнуть (что заранее также неизвестно и неочевидно) даже однопараметрическим анзацем

$$e^{i\nu\hat{\sigma}_{12}}|00\rangle = (\cos\nu\hat{\sigma}_{00} + i\sin\nu\hat{\sigma}_{12})|00\rangle = \cos\nu|00\rangle + \sin\nu|11\rangle,$$

с теми же значениями  $\cos\nu$  и  $\sin\nu$ , что и в (1.22).

Действуя на (1.21) оператором  $e^{i\xi\hat{\sigma}_{02}}$ , получим вектор состояния

$$\begin{aligned}
|\Phi\rangle &= \hat{U}(\boldsymbol{\theta})|00\rangle \\
&= (\cos\xi\hat{\sigma}_{00} + i\sin\xi\hat{\sigma}_{02})(e^{i(\mu+\lambda)}\cos\nu|00\rangle + e^{i(\mu+\pi/2-\lambda)}\sin\nu|11\rangle) \\
&= e^{i(\mu+\lambda)}\sin\xi\cos\nu|01\rangle - e^{i(\mu+\pi/2-\lambda)}\sin\xi\sin\nu|10\rangle \\
&\quad + e^{i(\mu+\lambda)}\cos\xi\cos\nu|00\rangle + e^{i(\mu+\pi/2-\lambda)}\cos\xi\sin\nu|11\rangle, \quad (1.23)
\end{aligned}$$



который зависит от четырех параметров. Из формы данного вектора видно, что анзац (1.20) универсален (с учетом замечания о вещественности коэффициентов, сделанного выше). Основное состояние достигается при произвольном  $\mu \in \mathbb{R}$  и

$$\xi = 0, \lambda = \{\pi/4, 7\pi/4\}, \cos\nu = 1/\sqrt{5}, \sin\nu = \{2/\sqrt{5}, -2/\sqrt{5}\} \quad (1.24)$$

или

$$\xi = \pi, \lambda = \{\pi/4, 7\pi/4\}, \cos\nu = -1/\sqrt{5}, \sin\nu = \{-2/\sqrt{5}, 2/\sqrt{5}\}. \quad (1.25)$$

Для сокращения записи имеет смысл освободиться в (1.23) от фазового множителя и записать вектор состояния в виде

$$\begin{aligned} |\Phi\rangle = \sin\xi \big( \cos\nu |01\rangle - e^{i(\pi/2-2\lambda)} \sin\nu |10\rangle \big) \\ + \cos\xi \big( \cos\nu |00\rangle + e^{i(\pi/2-2\lambda)} \sin\nu |11\rangle \big) \end{aligned} \quad (1.26)$$

*Второй шаг — вычисление энергии состояния*, т.е. среднего значения  $\langle\Phi|\hat{H}|\Phi\rangle$ . Заметим, что первый и второй шаги должны выполняться на квантовых устройствах, а при классической симуляции алгоритма необходимо проводить явные вычисления. Из (1.15) и (1.26) находим

$$\begin{aligned} \hat{H}|\Phi\rangle &= \sin\xi \big( 2\hat{\sigma}_{03} + \hat{\sigma}_{30} - 4\hat{\sigma}_{11} \big) \big( \cos\nu |01\rangle - e^{i(\pi/2-2\lambda)} \sin\nu |10\rangle \big) \\ &+ \cos\xi \big( 2\hat{\sigma}_{03} + \hat{\sigma}_{30} - 4\hat{\sigma}_{11} \big) \big( \cos\nu |00\rangle + e^{i(\pi/2-2\lambda)} \sin\nu |11\rangle \big) \\ &= \sin\xi \left\{ \left( 4e^{i(\pi/2-2\lambda)} \sin\nu - \cos\nu \right) |01\rangle \right. \\ &\quad \left. - \left( e^{i(\pi/2-2\lambda)} \sin\nu + 4\cos\nu \right) |10\rangle \right\} \\ &+ \cos\xi \left\{ \left( 3\cos\nu - 4e^{i(\pi/2-2\lambda)} \sin\nu \right) |00\rangle \right. \\ &\quad \left. - \left( 4\cos\nu + 3e^{i(\pi/2-2\lambda)} \sin\nu \right) |11\rangle \right\}. \end{aligned}$$

Поскольку

$$\begin{aligned} \langle\Phi| &= \sin\xi \big( \cos\nu \langle 01| - e^{-i(\pi/2-2\lambda)} \sin\nu \langle 10| \big) \\ &+ \cos\xi \big( \cos\nu \langle 00| + e^{-i(\pi/2-2\lambda)} \sin\nu \langle 11| \big), \end{aligned}$$

то

$$\begin{aligned}
E_{\Phi} &= \langle \Phi | \hat{H} | \Phi \rangle \\
&= \sin^2 \xi \left\{ \cos \nu (4e^{i(\pi/2-2\lambda)} \sin \nu - \cos \nu) \right. \\
&\quad \left. + \sin \nu (\sin \nu + 4e^{-i(\pi/2-2\lambda)} \cos \nu) \right\} \\
&\quad + \cos^2 \xi \left\{ \cos \nu (3 \cos \nu - 4e^{i(\pi/2-2\lambda)} \sin \nu) \right. \\
&\quad \left. - \sin \nu (4e^{-i(\pi/2-2\lambda)} \cos \nu + 3 \sin \nu) \right\} \\
&= \sin^2 \xi (4 \sin 2\lambda \sin 2\nu - \cos 2\nu) + \cos^2 \xi (3 \cos 2\nu - 4 \sin 2\lambda \sin 2\nu). \quad (1.27)
\end{aligned}$$

Разумеется, если взять значения  $\xi$ ,  $\lambda$ ,  $\cos \nu$ ,  $\sin \nu$  как в (1.24) или в (1.25), то мы получим энергию основного состояния (1.16), т.е.  $E_{\Phi} = -5$ .

*Третий шаг — изменение значений параметров  $\lambda, \mu, \nu$  (с целью минимизации  $E_{\Phi}$ ) и возвращение к первому шагу; предполагается, что в начале выполнения алгоритма начальные значения параметров заданы. Из (1.27) видно, что на значение  $E_{\Phi}$  параметр  $\mu$  не влияет, а параметры  $\lambda, \nu$  должны варьироваться в области  $[0, \pi] \times [0, \pi]$ . Однако изначально это неизвестно, поэтому все четыре параметра должны варьироваться в области  $[0, 2\pi] \times [0, 2\pi] \times [0, 2\pi] \times [0, 2\pi]$ . Имеет смысл установить независимость энергии от параметра  $\mu$  (и ее зависимость от остальных параметров) в начале работы алгоритма.*

Мы уже знаем, что глобальный минимум энергии достигается для четырех наборов параметров (1.24) и (1.25). Соответствующие собственные векторы, вычисленные по выражению (1.26) отличаются от (1.16) только фазовыми множителями. Теперь необходимо выяснить, имеются ли у функции (трех переменных) (1.27) другие локальные минимумы.

Используя систему Maple, вычислим производные

$$\begin{aligned}
&\partial_{\xi} E_{\Phi}, \quad \partial_{\lambda} E_{\Phi}, \quad \partial_{\nu} E_{\Phi}, \\
&A = \partial_{\xi}^2 E_{\Phi}, \quad B = \partial_{\lambda}^2 E_{\Phi}, \quad C = \partial_{\nu}^2 E_{\Phi}, \\
&K = \partial_{\xi\lambda} E_{\Phi}, \quad L = \partial_{\xi\nu} E_{\Phi}, \quad M = \partial_{\lambda\nu} E_{\Phi}.
\end{aligned}$$

Находим

$$\begin{aligned}
\partial_\xi E_\Phi &= 4 \sin 2\xi (2 \sin 2\lambda \sin 2\nu - \cos 2\nu), \\
\partial_\lambda E_\Phi &= -8 \cos 2\xi \cos 2\lambda \sin 2\nu, \\
\partial_\nu E_\Phi &= \sin^2 \xi (8 \sin 2\lambda \cos 2\nu + 2 \sin 2\nu) - \cos^2 \xi (6 \sin 2\nu + 8 \sin 2\lambda \cos 2\nu), \\
A &= 8 \cos 2\xi (2 \sin 2\lambda \sin 2\nu - \cos 2\nu), \\
B &= 16 \cos 2\xi \sin 2\lambda \sin 2\nu, \\
C &= 4 \cos^2 \xi (4 \sin 2\lambda \sin 2\nu - 3 \cos 2\nu) - 4 \sin^2 \xi (4 \sin 2\lambda \sin 2\nu - \cos 2\nu), \\
K &= 16 \sin 2\xi \cos 2\lambda \sin 2\nu, \\
L &= 4 \sin 2\xi (4 \sin 2\lambda \cos 2\nu + 2 \sin 2\nu), \\
M &= -16 \cos 2\xi \cos 2\lambda \cos 2\nu.
\end{aligned}$$

Необходимые и достаточные условия минимума имеют вид

$$\begin{aligned}
&\partial_\xi E_\Phi = 0, \quad \partial_\lambda E_\Phi = 0, \quad \partial_\nu E_\Phi = 0, \\
&A > 0, \quad \det \begin{pmatrix} A & K \\ K & B \end{pmatrix} > 0, \quad \det \begin{pmatrix} A & K & L \\ K & B & M \\ L & M & C \end{pmatrix} > 0.
\end{aligned}$$

Снова проводя вычисления с помощью системы Maple, обнаруживаем четыре точки локального минимума с энергией  $E_\Phi = -\sqrt{17}$ :

$$\begin{aligned}
\xi &= \frac{\pi}{2}, & \lambda &= \frac{\pi}{4}, & \nu &= \pi - \frac{1}{2} \arctan(4), \\
\xi &= \frac{\pi}{2}, & \lambda &= \frac{\pi}{4}, & \nu &= 2\pi - \frac{1}{2} \arctan(4), \\
\xi &= \frac{\pi}{2}, & \lambda &= \frac{3\pi}{4}, & \nu &= \frac{1}{2} \arctan(4), \\
\xi &= \frac{\pi}{2}, & \lambda &= \frac{3\pi}{4}, & \nu &= \pi + \frac{1}{2} \arctan(4).
\end{aligned}$$

Таким образом, в процессе оптимизации целевой функции должны использоваться методы, которые позволяют избежать попадания в точку локального минимума, например, метод отжига.

## 1.5 Оптимизация

## Глава 2

# Вариационный квантовый алгоритм на основе метода отжига

### 2.1 Метод отжига

Метод отжига, как фундаментальная концепция в решении задач глобальной оптимизации, находит широкое применение в квантовых вычислениях, особенно в контексте вариационных квантовых алгоритмов. Основная идея метода заключается в постепенном снижении "температуры" системы, чтобы достичь состояния минимальной энергии. В этом разделе подробно рассмотрим как классический, так и квантовый подходы к отжигу, их теоретические основы и практическое применение.

Классический метод отжига основывается на аналогии с физическим процессом термического отжига, при котором материал медленно охлаждается, чтобы избежать образования дефектов и достичь состояния минимальной энергии. Математическое основание метода связано с распределением Больцмана, которое описывает вероятность состояния системы при заданной температуре  $T$ :

$$p(x) = \frac{1}{Z(T)} \exp \left( -\frac{E(x)}{k_B T} \right), \quad (2.1)$$

где  $E(x)$  — энергия состояния  $x$ ,  $k_B$  — константа Больцмана, а  $Z(T)$  — статистическая сумма. Процесс отжига моделирует систему, которая может переходить между состояниями  $x$  и  $y$  с вероятностью, зависящей от разности энергий  $\Delta E = E(y) - E(x)$ :

$$p(x \rightarrow y) = \min \left( 1, \exp \left( -\frac{\Delta E}{k_B T} \right) \right). \quad (2.2)$$

С течением времени, температура  $T$  постепенно уменьшается, что приводит к уменьшению вероятности перехода в состояния с более высокой энергией, в то время как система стремится к состоянию глобального минимума энергии.

Квантовый алгоритм отжига использует преимущества квантовой механики, такие как суперпозиция и туннелирование, для более эффективного поиска глобального минимума. В отличие от классического подхода, квантовый отжиг позволяет системе преодолевать энергетические барьеры, используя когерентное туннелирование, что значительно увеличивает вероятность нахождения глобального минимума.

Квантовый отжиг моделируется с помощью временного гамильтониана, который постепенно изменяется от начального состояния к целевому:

$$H(t) = (1 - s(t))H_B + s(t)H_P, \quad (2.3)$$

где  $H_B$  — начальный гамильтониан, часто представляющий собой простую задачу, такую как сумма операторов Паули  $X$ , а  $H_P$  — проблемаспецифический гамильтониан. Функция  $s(t)$ , изменяющаяся от 0 до 1, управляет эволюцией системы от начального состояния к состоянию минимальной энергии.

Эволюция квантовой системы описывается уравнением Шрёдингера:

$$i\hbar \frac{\partial}{\partial t} |\psi(t)\rangle = H(t) |\psi(t)\rangle, \quad (2.4)$$

где  $\hbar$  — приведённая постоянная Планка. Это уравнение описывает, как квантовая система изменяется со временем под воздействием изменяющегося гамильтониана.

Важным аспектом квантового отжига является адъективная эволюция системы, которая позволяет системе оставаться в состоянии минимальной энергии в течение всего процесса. Это достигается за счёт медленного изменения параметра  $s(t)$  в соответствии с адъективной теоремой:

$$\frac{ds}{dt} \ll \frac{\Delta^2}{\hbar \left\| \frac{dH}{ds} \right\|}, \quad (2.5)$$

где  $\Delta$  — энергетический разрыв между основным и первым возбужденными состояниями гамильтониана.

## 2.2 Алгоритм

### 2.2.1 Этап 1: Чтение и парсинг данных гамильтониана

Гамильтониан представляется как линейная комбинация:

$$\hat{H} = \sum_{k=1}^K c_k \bigotimes_{j=1}^N \sigma_{k_j}^{(j)} \quad (2.6)$$

где  $\sigma_{k_j} \in \{I, X, Y, Z\}$  - операторы Паули для  $j$ -го кубита.

```

1 def read_hamiltonian_data(file_path):
2     """
3     Читает данные из файла hamiltonian_operators.txt и возвращает два списка:
4     - Список операторов Паули в виде коэффициента оператора и строки Паули.
5     - Список строк операторов Паули.
6
7     :param file_path: Путь к файлу.
8     :return: (pauli_operators, pauli_strings)
9     :raises FileNotFoundError: Если файл не найден.
10    """
11    lines = read_file_lines(file_path, ignore_comments=False)
12    pauli_operators = []
13    pauli_strings = []
14    for line in lines:
15        parts = line.strip().split()
16        if len(parts) == 3:

```

```

17         real_part, imag_part, index_str = (
18             float(parts[0]),
19             float(parts[1]),
20             str(parts[2]),
21         )
22         coefficient = np.complex128(real_part + imag_part * 1j)
23         index_list = [int(c) for c in index_str]
24         if coefficient != 0:
25             pauli_operators.append((coefficient, index_list))
26             pauli_strings.append(index_list)
27     return pauli_operators, pauli_strings

```

- `read_file_lines`: Чтение строк из файла с фильтрацией комментариев
- Преобразование строки вида "2.0 0.0 03" в:
  - Коэффициент:  $2.0 + 0.0i$
  - Оператор:  $\sigma_0 \otimes \sigma_3$
- Валидация данных через проверку `abs(coefficient) > 1e-12`

## 2.2.2 Этап 2: Инициализация параметров

Начальные значения  $\theta$  вычисляются по формуле:

$$\theta_m^{(0)} = \underbrace{\frac{2\pi|c_m|}{\max_j |c_j|}}_{\text{Нормировка}} + \underbrace{\mathcal{N}(0, 0.1)}_{\text{Шум}} \quad (2.7)$$

```

1 def generate_shifted_theta(pauli_operators: List[Tuple[complex,
2     List[int]]]) -> np.ndarray:
3     """Генерирует  $\theta$  на основе коэффициентов операторов (вектор сдвига)"""
4     if not pauli_operators:
5         return np.array([], dtype=np.float64)
6
7     # Извлечение модулей действительных частей
8     coeffs = np.array([abs(op[0].real) for op in pauli_operators],
9         dtype=np.float64)
10    norm = np.linalg.norm(coeffs)
11    if norm < 1e-12:

```

```

10     return np.zeros(len(coeffs))
11
12     # Масштабируем в [0, 2π) и добавляем шум
13     scaled = (coeffs / norm) * 2 * np.pi
14     return scaled + np.random.normal(0, 0.1, len(scaled))

```

- Использование `op[0].real` - акцент на действительной части коэффициентов
- Защита от деления на ноль через тернарный оператор
- Нормировка в диапазон  $[0, 2\pi)$  для соответствия периодичности экспоненты

### 2.2.3 Этап 3: Вычисление энергии

Вычисление энергии происходит с использованием формулы:

$$E(\theta) = \langle 0 | U^\dagger(\theta) \hat{H} U(\theta) | 0 \rangle = \sum_{k=1}^K c_k \langle 0 | U^\dagger P_k U | 0 \rangle \quad (2.8)$$

1. Построение анзаца (`calculate_ansatz`):

```

1 def calculate_ansatz(
2     theta: np.ndarray, pauli_operators: List[Tuple[complex,
3         List[int]]]
4 ) -> Tuple[Dict[Tuple[int, ...], complex], str, str]:
5     """
6     Вычисляет анзац в виде произведения экспонент операторов Паули.
7     Возвращает: (словарь операторов, символьное представление, числен-
8         ное представление)
9     """
10
11     operator_length = len(pauli_operators[0][1])
12     result = {tuple([0] * operator_length): 1.0}
13
14     for t, (_, op) in zip(theta, pauli_operators):
15         cos_t = np.cos(t)
16         sin_t = np.sin(t)
17         new_result: Dict[Tuple[int, ...], complex] = {}
18
19         for existing_op, existing_coeff in result.items():
20             # Слагаемое с cos(θ)*I
21             identity_coeff = existing_coeff * cos_t

```



```

19         new_result[existing_op] = new_result.get(existing_op,
20                                                    0) + identity_coeff
21
22         # Слагаемое с  $i \cdot \sin(\theta) \cdot \sigma$ 
23         pauli_coeff = existing_coeff * 1j * sin_t
24         compose_coeff, compose_op =
25             pauli_compose(list(existing_op), op)
26         final_coeff = pauli_coeff * compose_coeff
27         final_op = tuple(compose_op)
28         new_result[final_op] = new_result.get(final_op, 0) +
29             final_coeff
30
31     result = new_result
32
33     symbolic_str, numeric_str = format_ansatz(pauli_operators,
34                                              result)
35     return result, symbolic_str, numeric_str

```

## 2. Композиция операторов (pauli\_compose):

```

1 def pauli_compose(s1: List[int], s2: List[int]) -> Tuple[complex,
2 List[int]]:
3     """
4     Вычисляет композицию двух операторов Паули.
5     Возвращает: (коэффициент, результирующий оператор)
6     """
7     coefficient = 1.0
8     result = []
9     for a, b in zip(s1, s2):
10         coeff, idx = multiply_pauli(a, b)
11         coefficient *= coeff
12         result.append(idx)
13     return coefficient, result

```

### 2.2.4 Этап 4: Алгоритм отжига

Сам алгоритм отжига описан в данном коде

```

1 def simulated_annealing(
2     initial_theta: np.ndarray,
3     pauli_operators: list,
4     initial_temp: float = 1000.0,
5     cooling_rate: float = 0.99,
6     min_temp: float = 1e-5,
7     num_iterations_per_temp: int = 500,

```

```

8     step_size: float = 0.5,
9 ) -> tuple:
10     """Алгоритм отжига с термализацией."""
11     current_theta = initial_theta.copy()
12     best_theta = current_theta.copy()
13     best_energy = float("inf")
14     rng = np.random.default_rng()
15
16     with Progress(
17         SpinnerColumn(),
18         TextColumn("[progress.description]{task.description}"),
19         BarColumn(bar_width=None),
20         TextColumn("[progress.percentage]{task.percentage:>3.0f}%"),
21     ) as progress:
22         task = progress.add_task("[cyan]Отжиг...", total=100)
23
24         temp = initial_temp
25         iteration = 0
26         thermalization_steps = int(num_iterations_per_temp * 0.2) #
27                                20% шагов на термализацию
28
29         while temp > min_temp:
30             # Фаза термализации
31             for _ in range(thermalization_steps):
32                 current_theta = generate_shifted_theta(pauli_operators)
33                 ansatz_dict, _, _ = calculate_ansatz(current_theta,
34                                                         pauli_operators)
35                 uhu_dict = compute_uhu(ansatz_dict, pauli_operators)
36                 current_energy = calculate_expectation(uhu_dict)
37
38                 if current_energy < best_energy:
39                     best_theta = current_theta.copy()
40                     best_energy = current_energy
41
42             # Основная фаза отжига
43             for _ in range(num_iterations_per_temp):
44                 perturbation = rng.normal(0,
45                                         step_size*(temp/initial_temp), current_theta.shape)
46                 neighbor_theta = (current_theta + perturbation) %
47                                 (2*np.pi)
48
49                 ansatz_dict, _, _ = calculate_ansatz(neighbor_theta,
50                                                         pauli_operators)
51                 uhu_dict = compute_uhu(ansatz_dict, pauli_operators)
52                 current_energy = calculate_expectation(uhu_dict)
53
54                 energy_diff = current_energy - best_energy
55                 if energy_diff < 0 or rng.random() <
56                     np.exp(-energy_diff / temp):
57                     current_theta = neighbor_theta.copy()

```

```

53         if current_energy < best_energy:
54             best_theta = current_theta.copy()
55             best_energy = current_energy
56
57         iteration += 1
58         if iteration % 100 == 0:
59             progress.update(task, advance=1)
60
61         temp *= cooling_rate
62
63     return best_theta, best_energy

```

## Ключевые особенности реализации

- Адаптивный размер шага:  $\sigma(T) = \sigma_0 \cdot T/T_{\text{init}}$
- Двухэтапный процесс: термализация + основной цикл
- Сохранение лучшего решения в отдельной переменной
- Модульная структура с возможностью изменения:
  - `thermalization_steps` - число шагов термализации
  - `iter_per_temp` - итераций на одну температуру
  - `cooling_rate` - скорость охлаждения  $\alpha$
- Использование `defaultdict` для эффективного управления операторами
- Стабилизация численных вычислений через отсечение малых значений ( $< 10^{-12}$ )
- Векторизованные операции с NumPy для ускорения вычислений
- Логирование процесса оптимизации в `output.log`

## 2.3 Сравнительные результаты тестирования

# Заключение

# Литература

- [1] V. V. Nikonov, A. N. Tsirulev. *Pauli basis formalism in quantum computations*. Volume 8, No 3, pp. 1 – 14, 2020.  
([doi:10.26456/mmg/2020-831](#))
- [2] J. Preskill. *Quantum Computing in the NISQ era and beyond*. Quantum, vol. 2, p. 79, 2018.  
([quantum-journal:q-2018-08-06-79](#))
- [3] M. Cerezo, et al. *Variational Quantum Algorithms*. Nature Reviews Physics, vol. 3, pp. 625-644, 2021.  
([nature:42254-021-00348-9](#))
- [4] A. Peruzzo, et al. *A variational eigenvalue solver on a photonic quantum processor*. Nature Communications, vol. 5, p. 4213, 2014.  
([nature:ncomms5213](#))
- [5] E. Farhi, J. Goldstone, and S. Gutmann. *A Quantum Approximate Optimization Algorithm*. arXiv preprint arXiv:1411.4028, 2014.  
([arXiv:1411.4028](#))
- [6] J. R. McClean, et al. *The theory of variational hybrid quantum-classical algorithms*. New Journal of Physics, vol. 18, p. 023023, 2016.  
([iopscience:1367-2630-18-2-023023](#))
- [7] A. Kandala, et al. *Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets*. Nature, vol. 549, pp. 242-246, 2017.  
([nature:nature23879](#))
- [8] A. W. Harrow, A. Hassidim, and S. Lloyd. *Quantum algorithm for linear systems of equations*. Physical Review Letters, vol. 103, no. 15, p. 150502,

2009.  
([aps:PhysRevLett.103.150502](#))
- [9] J. Biamonte, et al. *Quantum machine learning*. Nature, vol. 549, pp. 195-202, 2017.  
([nature:nature23474](#))
- [10] A. A. Lopatin. *Квантовая механика и её приложения*. Санкт-Петербургский Государственный Университет.  
([math.spbu:user/gran/sb1/lopatin](#))
- [11] A. Aspuru-Guzik, A. D. Dutoi, P. J. Love, M. Head-Gordon. *Simulated Quantum Computation of Molecular Energies*. Science, vol. 309, no. 5741, pp. 1704-1707, 2005.  
([science:1113479](#))
- [12] M. Schuld, I. Sinayskiy, F. Petruccione. *An introduction to quantum machine learning*. Contemporary Physics, vol. 56, no. 2, pp. 172-185, 2015.  
([tandfonline:00107514.2014.964942](#))
- [13] A. Daskin, S. Kais. *Decomposition of unitary matrices for finding quantum circuits: Application to molecular Hamiltonians*. The Journal of Chemical Physics, vol. 141, no. 23, p. 234115, 2014.  
([aip:1.4904315](#))
- [14] J. Romero, R. Babbush, J. R. McClean, C. Hempel, P. J. Love, A. Aspuru-Guzik. *Strategies for quantum computing molecular energies using the unitary coupled cluster ansatz*. Quantum Science and Technology, vol. 4, no. 1, p. 014008, 2018.  
([iopscience:2058-9565/aad3e4](#))
- [15] V. Havlicek, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, J. M. Gambetta. *Supervised learning with quantum-enhanced feature spaces*. Nature, vol. 567, pp. 209-212, 2019.  
([nature:s41586-019-0980-2](#))
- [16] N. Moll, P. Barkoutsos, L. Bishop, J. M. Chow, A. Cross, D. J. Egger, S. Filipp, A. Fuhrer, J. M. Gambetta, M. Ganzhorn, et al. *Quantum*

*optimization using variational algorithms on near-term quantum devices.*  
Quantum Science and Technology, vol. 3, no. 3, p. 030503, 2018.  
([iopscience:2058-9565/aab822](#))

# Приложения

## Приложение А

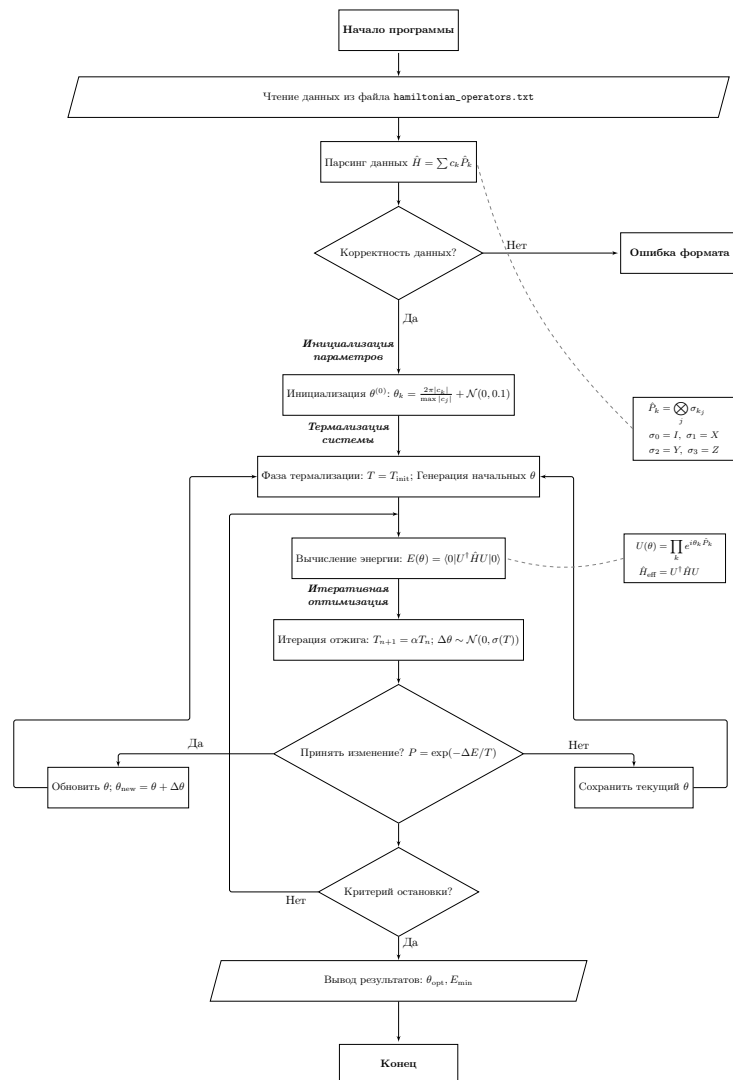


Рис. 2.1: Блок-схема алгоритма



# Приложение Python

```
1 #constants/file_paths.py
2 import sys
3 from pathlib import Path
4
5 def get_base_path() -> Path:
6     """Возвращает путь к директории с EXE (или к проекту в
7     dev-режиме)."""
8     if getattr(sys, "frozen", False):
9         # Для скомпилированного EXE берём директорию исполняемого файла
10         return Path(sys.argv[0]).parent
11     else:
12         # Для разработки возвращаем корень проекта
13         return Path(__file__).parent.parent
14
15 # Путь к hamiltonian_operators.txt (внешний файл)
16 HAMILTONIAN_FILE_PATH = get_base_path() / "params" /
17     "hamiltonian_operators.txt"
18
19 # Файл вывода создаётся в текущей рабочей директории
20 OUTPUT_FILE_PATH = get_base_path() / "output.log"
21
22 #constants/pauli.py
23 PAULI_MAP = {
24     (1, 2): (1j, 3),
25     (2, 1): (-1j, 3),
26     (3, 1): (1j, 2),
27     (1, 3): (-1j, 2),
28     (2, 3): (1j, 1),
29     (3, 2): (-1j, 1),
30 }
31
32 #params/hamiltonian_operators.txt
33 # Действительная часть | Мнимая часть | Строка Паули
34
35 2.0 0.0 03
36 1.0 0.0 30
37 -4.0 0.0 11
38
39
40 #utils/console_and_print.py
41 from rich.console import Console
42 from typing import Any
43 from constants.file_paths import OUTPUT_FILE_PATH
44
45
46 def console_and_print(console: Console, message: Any) -> None:
47     """
```

```

48     Выводит результат выполнения программы в файл и консоль
49
50     :param console: Объект Console для вывода в консоль
51     :param message: Сообщение для вывода
52     """
53     console.print(message)
54     with open(OUTPUT_FILE_PATH, "a", encoding="utf-8") as file:
55         file.write(console.export_text() + "\n")
56
57
58 #utils/create_table.py
59 from rich.table import Table
60 from rich.panel import Panel
61 from rich import box
62 from typing import List, Dict, Any
63
64
65 def create_table(
66     columns: List[Dict[str, str]],
67     data: List[List[Any]],
68     title: str,
69     border_style: str = "yellow",
70 ) -> Panel:
71     table = Table(box=box.ROUNDED, border_style="yellow")
72     for col in columns:
73         table.add_column(
74             col["name"],
75             justify=col.get("justify", "default"),
76             style=col.get("style", ""),
77         )
78     for row in data:
79         table.add_row(*row)
80     return Panel(table, title=title, border_style=border_style)
81
82
83 #utils/format_ansatz.py
84 from .format_complex_number import format_complex_number
85
86
87 def format_ansatz(
88     pauli_operators: list[list[int]], result: dict[tuple, complex]
89 ) -> tuple[str, str]:
90     """
91     Форматирует анзац в символьное и численное представление.
92
93     :param pauli_operators: Список операторов Паули (списки индексов).
94     :param result: Словарь с результатами вычислений {оператор: коэффициент}.
95     :return: Символьное представление анзаца и его численное значение.
96     """
97     # Формируем  $U(\theta)$ 

```

```

98     ansatz_symbolic = "U( $\theta$ ) = " + " * ".join(
99         [
100             f"e^{i\theta_{i+1}}*{format_complex_number(op[0])}* $\sigma_{\{'\}}$ ".join(map(str,
101                 op[1:]))})"
102             for i, op in enumerate(pauli_operators)
103         ]
104     )
105     # Формируем U
106     ansatz_numeric = "U = " + " + ".join(
107         [
108             f"{format_complex_number(c)}*{format_complex_number(op[0])}* $\sigma_{\{'\}}$ ".join(
109                 op[1:]))})"
110             if isinstance(op, (list, tuple))
111             and len(op) > 1
112             and isinstance(op[0], complex)
113             else f"{format_complex_number(c)}* $\sigma_{\{'\}}$ ".join(map(str, op))}"
114             for op, c in result.items()
115         ]
116     )
117     return ansatz_symbolic, ansatz_numeric
118
119
120 #utils/format_complex_number.py
121 from sympy import re as sp_re, im as sp_im
122 from .format_number import format_number
123
124
125 def format_complex_number(c) -> str:
126     """
127     Универсальный форматировщик комплексных чисел для SymPy и стандартны
128     x типов.
129     """
130     # Извлечение компонентов через SymPy
131     real = float(sp_re(c))
132     imag = float(sp_im(c))
133
134     real_str = format_number(real) if abs(real) > 1e-12 else ""
135     imag_str = ""
136
137     if abs(imag) > 1e-12:
138         abs_imag = abs(imag)
139         imag_value = format_number(abs_imag)
140
141         # Специальные случаи для  $\pm 1$ 
142         if imag_value == "1":
143             imag_str = "i" if imag > 0 else "-i"
144         else:
145             imag_sign = "" if imag > 0 else "-"
146             imag_str = f"{imag_sign}{imag_value}i"

```

```

146
147     # Сборка результата
148     parts = []
149     if real_str:
150         parts.append(real_str)
151     if imag_str:
152         parts.append(imag_str)
153
154     if not parts:
155         return "0"
156
157     # Корректное объединение компонентов
158     result = parts[0]
159     for part in parts[1:]:
160         if part.startswith("-"):
161             result += f"-{part[1:]}"
162         else:
163             result += f"+{part}"
164
165     # Фикс артефактов форматирования
166     return result.replace("+ -", "- ").replace("1i",
167         "i").replace(".0i", "i")
168
169 #utils/format_number.py
170 def format_number(num: float | int) -> str:
171     """
172     Универсальный форматировщик чисел с точным определением целочисленны
173     х значений
174     и подавлением артефактов вычислений с плавающей точкой.
175     """
176     # Проверка на целое число с учетом погрешности вычислений
177     if abs(num - round(num)) < 1e-15:
178         return str(int(round(num)))
179
180     # Форматирование с ручным управлением десятичными знаками
181     s = f"{num:.14f}".rstrip("0").rstrip(".")
182
183     # Исправление артефактов вида ".5" → "0.5"
184     if s.startswith("."):
185         s = "0" + s
186     elif s.startswith("-."):
187         s = s.replace("-.", "-0.")
188
189     # Удаление лишних десятичных знаков после 4-го
190     if "." in s:
191         int_part, dec_part = s.split(".")
192         dec_part = dec_part[:4].ljust(4, "0").rstrip("0")
193         s = f"{int_part}.{dec_part}" if dec_part else int_part
194
195     return s

```

```

195
196
197 #utils/get_operator_for_console.py
198 from .format_complex_number import format_complex_number
199 from typing import Union
200
201
202 def get_operator_for_console(c: Union[complex, float, int], i: str) ->
    str:
203     """
204     Функция создана для 'красивого' вывода оператора Паули в консоль.
205     Избегает ситуаций, когда у нас выводится конструкция вида '1*σ'
206
207     :param c: Коэффициент оператора Паули.
208     :param i: Строка оператора Паули.
209     :return: Отформатированную строку.
210     """
211     if c == 1:
212         return f"σ_{i}"
213     else:
214         return f"{format_complex_number(c)}*σ_{i}"
215
216
217 #utils/initialize_environment.py
218 from rich.console import Console # Для красивого вывода в консоль
219 from constants.file_paths import OUTPUT_FILE_PATH
220
221
222 def initialize_environment() -> Console:
223     """Инициализирует окружение и возвращает консольный объект."""
224     if OUTPUT_FILE_PATH.exists():
225         OUTPUT_FILE_PATH.unlink()
226     return Console(force_terminal=True, color_system="truecolor",
227                   record=True)
228
229 #utils/print_composition_table.py
230 from rich.console import Console # Для красивого вывода в консоль
231 from typing import Tuple, List, Callable
232 from .console_and_print import console_and_print
233 from .create_table import create_table
234 from .format_complex_number import format_complex_number
235
236
237 def print_composition_table(
238     console: Console,
239     pauli_compose: Callable[[List[int], List[int]], Tuple[complex,
240         List[int]]],
241     pauli_strings: List[List[int]],
242 ) -> None:
243     """Выводит таблицу композиций операторов Паули."""

```

```

243     results = []
244     for s1 in pauli_strings:
245         for s2 in pauli_strings:
246             coeff, product = pauli_compose(s1, s2)
247             results.append((s1, s2, format_complex_number(coeff),
248                             product))
249
250     table_data = [
251         [str(s1), str(s2), str(h).lower(), str(p)] for s1, s2, h, p in
252         results
253     ]
254     console_and_print(
255         console,
256         create_table(
257             columns=[
258                 {"name": "Оператор 1", "style": "cyan", "justify":
259                 "center"},
260                 {"name": "Оператор 2", "style": "magenta", "justify":
261                 "center"},
262                 {"name": "Коэффициент", "style": "green", "justify":
263                 "center"},
264                 {"name": "Результат", "style": "red", "justify":
265                 "center"},
266             ],
267             data=table_data,
268             title="Композиции операторов Паули",
269             border_style="green",
270         ),
271     )
272
273 #utils/print_hamiltonian.py
274 from rich.console import Console
275 from rich.panel import Panel
276 from typing import Tuple, List
277 from .get_operator_for_console import get_operator_for_console
278 from .console_and_print import console_and_print
279
280 def print_hamiltonian(
281     console: Console, pauli_operators: List[Tuple[complex, List[int]]]
282 ) -> None:
283     """Выводит представление гамильтониана в консоль."""
284     hamiltonian_str = "H = " + " + ".join(
285         [get_operator_for_console(c, ''.join(map(str, i))) for c, i in
286          pauli_operators]
287     )
288     console_and_print(
289         console,
290         Panel(
291             hamiltonian_str,

```

```

287         title="[bold]Введенный гамильтониан[/bold]",
288         border_style="green",
289     ),
290 )
291
292
293 #utils/print_pauli_table.py
294 from rich.console import Console # Для красивого вывода в консоль
295 from typing import Tuple, List
296 from .format_complex_number import format_complex_number
297 from .console_and_print import console_and_print
298 from .create_table import create_table
299
300
301 def print_pauli_table(
302     console: Console, pauli_operators: List[Tuple[complex, List[int]]]
303 ) -> None:
304     """Выводит таблицу операторов Паули."""
305     table_data = [[format_complex_number(c), str(i)] for c, i in
306                   pauli_operators]
307     console_and_print(
308         console,
309         create_table(
310             columns=[
311                 {"name": "Коэффициент", "style": "cyan"},
312                 {"name": "Индекс", "style": "magenta", "justify":
313                  "center"}],
314             data=table_data,
315             title="Операторы Паули",
316             border_style="purple",
317         ),
318     )
319
320 #utils/read_file_lines.py
321 from pathlib import Path
322
323
324 def read_file_lines(file_path, ignore_comments):
325     """
326     Читает строки из файла, игнорируя комментарии (строки, начинающиеся
327     с '#').
328
329     :param file_path: Путь к файлу.
330     :param ignore_comments: Игнорировать строки, начинающиеся с '#'.
331     :return: Список строк.
332     :raises FileNotFoundError: Если файл не найден.
333     """
334     file_path = Path(file_path) if not isinstance(file_path, Path) else
335         file_path

```

```

334     if not file_path.exists():
335         raise FileNotFoundError(f"Файл {file_path} не найден.")
336     with open(file_path, "r") as file:
337         return [
338             line.strip()
339             for line in file
340             if not (ignore_comments and line.strip().startswith("#"))
341         ]
342
343
344 #utils/read_hamiltonian_data.py
345 import numpy as np # Для работы с комплексными числами и математическим
    и операциями
346 from .read_file_lines import read_file_lines
347
348
349 def read_hamiltonian_data(file_path):
350     """
351     Читает данные из файла hamiltonian_operators.txt и возвращает два сп
    иска:
352     - Список операторов Паули в виде коэффициента оператора и строки Пау
    ли.
353     - Список строк операторов Паули.
354
355     :param file_path: Путь к файлу.
356     :return: (pauli_operators, pauli_strings)
357     :raises FileNotFoundError: Если файл не найден.
358     """
359     lines = read_file_lines(file_path, ignore_comments=False)
360     pauli_operators = []
361     pauli_strings = []
362     for line in lines:
363         # Разделение строки на компоненты
364         parts = line.strip().split()
365         if len(parts) == 3:
366             real_part, imag_part, index_str = (
367                 float(parts[0]),
368                 float(parts[1]),
369                 str(parts[2]),
370             )
371             # Преобразование в комплексное число
372             coefficient = np.complex128(real_part + imag_part * 1j)
373             # Парсинг строки Паули
374             index_list = [int(c) for c in index_str]
375             if coefficient != 0: # Игнорирование нулевых коэффициентов
376                 pauli_operators.append((coefficient, index_list))
377                 pauli_strings.append(index_list)
378     return pauli_operators, pauli_strings
379
380
381 #program.py

```



```

382 import numpy as np
383 import sys
384 import io
385 from rich.progress import Progress, BarColumn, TextColumn, SpinnerColumn
386 from rich.panel import Panel
387 from typing import Tuple, List, Dict
388
389 # Импорт самописных util функций
390 from utils.console_and_print import console_and_print
391 from utils.print_pauli_table import print_pauli_table
392 from utils.read_hamiltonian_data import read_hamiltonian_data
393 from utils.print_hamiltonian import print_hamiltonian
394 from utils.print_composition_table import print_composition_table
395 from utils.format_ansatz import format_ansatz
396 from utils.initialize_environment import initialize_environment
397
398 # Импорт констант
399 from constants.file_paths import HAMILTONIAN_FILE_PATH
400 from constants.pauli import PAULI_MAP
401
402 # Установка кодировки для корректного вывода
403 sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding="utf-8")
404 sys.stderr = io.TextIOWrapper(sys.stderr.buffer, encoding="utf-8")
405
406
407 def generate_shifted_theta(pauli_operators: List[Tuple[complex,
408     List[int]]]) -> np.ndarray:
409     """Генерирует  $\theta$  на основе коэффициентов операторов (вектор сдвига)."""
410     if not pauli_operators:
411         return np.array([], dtype=np.float64)
412
413     # Извлечение модулей действительных частей
414     coeffs = np.array([abs(op[0].real) for op in pauli_operators],
415         dtype=np.float64)
416     norm = np.linalg.norm(coeffs)
417     if norm < 1e-12:
418         return np.zeros(len(coeffs))
419
420     # Масштабируем в  $[0, 2\pi)$  и добавляем шум
421     scaled = (coeffs / norm) * 2 * np.pi
422     return scaled + np.random.normal(0, 0.1, len(scaled))
423
424 def multiply_pauli(i: int, j: int) -> Tuple[complex, int]:
425     """
426     Вычисляет произведение базисных операторов Паули.
427     Возвращает: (коэффициент, индекс результата)
428     """
429     if i == j:
430         return (1, 0)

```

```

430     if i == 0:
431         return (1, j)
432     if j == 0:
433         return (1, i)
434     return PAULI_MAP.get((i, j), (1, 0))
435
436
437 def pauli_compose(s1: List[int], s2: List[int]) -> Tuple[complex,
438 List[int]]:
439     """
440     Вычисляет композицию двух операторов Паули.
441     Возвращает: (коэффициент, результирующий оператор)
442     """
443     coefficient = 1.0
444     result = []
445     for a, b in zip(s1, s2):
446         coeff, idx = multiply_pauli(a, b)
447         coefficient *= coeff
448         result.append(idx)
449     return coefficient, result
450
451 def calculate_ansatz(
452     theta: np.ndarray, pauli_operators: List[Tuple[complex, List[int]]]
453 ) -> Tuple[Dict[Tuple[int, ...], complex], str, str]:
454     """
455     Вычисляет анзац в виде произведения экспонент операторов Паули.
456     Возвращает: (словарь операторов, символьное представление, численное
457     представление)
458     """
459     operator_length = len(pauli_operators[0][1])
460     result = {tuple([0] * operator_length): 1.0}
461
462     for t, (_, op) in zip(theta, pauli_operators):
463         cos_t = np.cos(t)
464         sin_t = np.sin(t)
465         new_result: Dict[Tuple[int, ...], complex] = {}
466
467         for existing_op, existing_coeff in result.items():
468             # Слагаемое с  $\cos(\theta) \cdot I$ 
469             identity_coeff = existing_coeff * cos_t
470             new_result[existing_op] = new_result.get(existing_op, 0) +
471                 identity_coeff
472
473             # Слагаемое с  $i \cdot \sin(\theta) \cdot \sigma$ 
474             pauli_coeff = existing_coeff * 1j * sin_t
475             compose_coeff, compose_op =
476                 pauli_compose(list(existing_op), op)
477             final_coeff = pauli_coeff * compose_coeff
478             final_op = tuple(compose_op)

```

```

476         new_result[final_op] = new_result.get(final_op, 0) +
            final_coeff
477
478     result = new_result
479
480     symbolic_str, numeric_str = format_ansatz(pauli_operators, result)
481     return result, symbolic_str, numeric_str
482
483
484 def compute_uhu(
485     u_dict: Dict[Tuple[int, ...], complex], h_terms:
486         List[Tuple[complex, List[int]]]
487 ) -> Dict[Tuple[int, ...], complex]:
488     """
489     Вычисляет оператор  $U^\dagger H U$ .
490     Возвращает: словарь {оператор: коэффициент}
491     """
492     uhu_dict: Dict[Tuple[int, ...], complex] = {}
493
494     for coeff_h, op_h in h_terms:
495         for j_op, j_coeff in u_dict.items():
496             conjugated_j_coeff = np.conj(j_coeff)
497             c1, op_uh = pauli_compose(list(j_op), op_h)
498
499             for k_op, k_coeff in u_dict.items():
500                 c2, op_uhu = pauli_compose(op_uh, list(k_op))
501                 total_coeff = conjugated_j_coeff * k_coeff * coeff_h *
502                     c1 * c2
503
504                 # Стабилизация малых значений
505                 if abs(total_coeff) < 1e-12:
506                     continue
507
508                 op_tuple = tuple(op_uhu)
509                 uhu_dict[op_tuple] = uhu_dict.get(op_tuple, 0) +
510                     total_coeff
511
512     return uhu_dict
513
514
515 def calculate_expectation(uhu_dict: Dict[Tuple[int, ...], complex]) ->
516     float:
517     """
518     Вычисляет  $\langle 0 | U^\dagger H U | 0 \rangle$  для состояния  $|0 \dots 0\rangle$ .
519     Возвращает: ожидаемое значение
520     """
521     expectation = 0.0
522     for op, coeff in uhu_dict.items():
523         if all(p in {0, 3} for p in op):
524             expectation += coeff.real
525     return expectation

```

```

522
523 def generate_neighbor_theta(
524     current_theta: np.ndarray, step_size: float = 0.1
525 ) -> np.ndarray:
526     """Генерирует соседнее решение, добавляя случайное изменение к текущ
        ему theta."""
527     perturbation = np.random.normal(scale=step_size,
        size=current_theta.shape)
528     return np.clip(current_theta + perturbation, 0.0, 1.0)
529
530
531 def simulated_annealing(
532     initial_theta: np.ndarray,
533     pauli_operators: list,
534     initial_temp: float = 1000.0,
535     cooling_rate: float = 0.99,
536     min_temp: float = 1e-5,
537     num_iterations_per_temp: int = 500,
538     step_size: float = 0.5,
539 ) -> tuple:
540     """Алгоритм отжига с термализацией."""
541     current_theta = initial_theta.copy()
542     best_theta = current_theta.copy()
543     best_energy = float("inf")
544     rng = np.random.default_rng()
545
546     with Progress(
547         SpinnerColumn(),
548         TextColumn("[progress.description]{task.description}"),
549         BarColumn(bar_width=None),
550         TextColumn("[progress.percentage]{task.percentage:>3.0f}%"),
551     ) as progress:
552         task = progress.add_task("[cyan]Отжиг...", total=100)
553
554         temp = initial_temp
555         iteration = 0
556         thermalization_steps = int(num_iterations_per_temp * 0.2) #
            20% шагов на термализацию
557
558         while temp > min_temp:
559             # Фаза термализации
560             for _ in range(thermalization_steps):
561                 current_theta = generate_shifted_theta(pauli_operators)
562                 ansatz_dict, _, _ = calculate_ansatz(current_theta,
                    pauli_operators)
563                 uhu_dict = compute_uhu(ansatz_dict, pauli_operators)
564                 current_energy = calculate_expectation(uhu_dict)
565
566                 if current_energy < best_energy:
567                     best_theta = current_theta.copy()
568                     best_energy = current_energy

```

```

569
570     # Основная фаза отжига
571     for _ in range(num_iterations_per_temp):
572         perturbation = rng.normal(0,
573             step_size*(temp/initial_temp), current_theta.shape)
574         neighbor_theta = (current_theta + perturbation) %
575             (2*np.pi)
576
577         ansatz_dict, _, _ = calculate_ansatz(neighbor_theta,
578             pauli_operators)
579         uhu_dict = compute_uhu(ansatz_dict, pauli_operators)
580         current_energy = calculate_expectation(uhu_dict)
581
582         energy_diff = current_energy - best_energy
583         if energy_diff < 0 or rng.random() <
584             np.exp(-energy_diff / temp):
585             current_theta = neighbor_theta.copy()
586
587             if current_energy < best_energy:
588                 best_theta = current_theta.copy()
589                 best_energy = current_energy
590
591             iteration += 1
592             if iteration % 100 == 0:
593                 progress.update(task, advance=1)
594
595         temp *= cooling_rate
596
597     return best_theta, best_energy
598
599
600 def main():
601     """Основная логика программы."""
602     console = initialize_environment()
603
604     if not HAMILTONIAN_FILE_PATH.exists():
605         msg = (
606             f"Файл [bold]{HAMILTONIAN_FILE_PATH}[/] не найден!\n"
607             "Убедитесь, что рядом с EXE есть папка [bold]params[/] с фай-
608             лом [bold]hamiltonian_operators.txt[/].")
609         console_and_print(console, Panel(msg, border_style="red"))
610         return
611
612     try:
613         pauli_operators, pauli_strings =
614             read_hamiltonian_data(HAMILTONIAN_FILE_PATH)
615         print_hamiltonian(console, pauli_operators)
616         print_pauli_table(console, pauli_operators)
617         print_composition_table(console, pauli_compose, pauli_strings)
618     except FileNotFoundError:

```

```

614     console_and_print(console, Panel(f"[red]Файл
        {HAMILTONIAN_FILE_PATH} не найден[/red]",
        border_style="red"))
615     return
616
617 if len(pauli_operators) < 2:
618     console_and_print(console, Panel("[red]Требуется минимум 2 опера
        топа Паули[/red]", border_style="red"))
619     return
620
621 best_energy = float("inf")
622 best_result = None
623 all_results = []
624
625 # Для каждого m используем первые m операторов
626 for m in range(2, len(pauli_operators) + 1):
627     current_ops = pauli_operators[:m]
628     initial_theta = generate_shifted_theta(current_ops) # Вектор сд
        вига!
629
630 # Оптимизация только для текущего подмножества операторов
631 optimized_theta, energy = simulated_annealing(
632     initial_theta=initial_theta,
633     pauli_operators=current_ops, # Передаем подмножество
634     initial_temp=100.0,
635     cooling_rate=0.95,
636     min_temp=1e-3,
637     num_iterations_per_temp=100,
638     step_size=0.1,
639 )
640
641 all_results.append({
642     "m": m,
643     "theta": optimized_theta,
644     "energy": energy,
645     "operators": current_ops
646 })
647
648 if energy < best_energy:
649     best_energy = energy
650     best_result = all_results[-1]
651
652 # Вывод результатов для лучшего набора параметров
653 _, ansatz_symbolic, ansatz_numeric = calculate_ansatz(
654     best_result["theta"],
655     best_result["operators"] # Используем сохраненное подмножество
656 )
657
658 console_and_print(console, Panel(ansatz_symbolic,
659     title="[bold]Символьное представление анзаца[/]",
        border_style="green"))

```

```

660
661     console_and_print(console, Panel(ansatz_numeric,
662         title="[bold]Численное представление анзаца[/]",
663         border_style="purple"))
664
665     console_and_print(console, Panel(f"{best_result['energy']:.6f}",
666         title="[bold]Энергия ( $\langle 0|U^\dagger H U|0 \rangle$  для состояния  $|0\dots 0\rangle$ )[/]",
667         border_style="green"))
668
669     input('Нажмите Enter для выхода...')
670
671 if __name__ == "__main__":
672     main()

```