Efficiently Trotterizing Lithium Hydride Time Eveloution Step Experimential Project Classiq

David Ponarovsky

July 29, 2023

Abstract

In this paper, we review our experimental implementation of an algorithm for designing time evolution simulation circuits of molecules, given their Hamiltonian in the Pauli basis. Our algorithm employs several heuristics to minimize the circuit depth. We demonstrated this by taking the lithium hydride molecule as a use case and comparing the generated circuit to the one produced by the naive approach.

1 Preamble.

Simulation is definitely one of the core ingredients that our modern civilization relies on; the fact that one can check on their computer if their sketch for an electronic chip or plane will actually compute or fly without the need to construct and test a physical prototype both cheapens and accelerates the development process.

Though humanity has successfully enabled an efficient simulation in a variety of regimes, there are still areas in which we don't have any other choices than performing physical experiments. Development of materials, drugs, and medicines is a good example of a processes that suffer from the lack of simulation and can last for decades. What distinguishes those areas is the fact that the mechanics theory governing them is quantum mechanics, which we have good reason to believe cannot be simulated by classical computers. Yet, quantum computers are considered to be good candidates to overcome that barrier.

Unfortunately, our current quantum machines are extremely limited in terms of computing resources; the most advanced computers have access to no more than a few hundred qubits and suffer from a considerable amount of noise, which cause long computations resulting in nonsense. Therefore, any short-term application must be cost-efficient.

On May 2022, Classiq, a pioneering quantum computing company, launched the first (at least at that scale)

competition in quantum circuits programming. The initial purpose of this work was to solve the Lithium simulation problem. In short, the problem asked to come up with a quantum circuit, restricted to only a few qubits, that progress the lithium hydride molecule in time. The winner of the competition would be the circuit with the shortest depth. We mention that our algorithm is generic and the software we have written can easily be modified for any other Hamiltonian.

The paper organized in the follow systematically manner, first in the preamble we will present the problem and talk about the general concepts which describe our specific Hamiltonian, including the quantities regime (i.e number of qubits vs number of Hamiltonians terms), the naive solution, and how mach improvement we could hope to make.

Then in next section we review all the techniques that we have used to improve a local gates assignment. Namely, this section deals with the question of how presenting each of the local term as subgate.

In contrast, the third and the forth sections reviews methods to determinate an order which clearly superior comparing to the naive approach. The forth section presenting an concept of analyzing the "product graph" a concept which could be thought as the second order analyses of the alternate path from the third section.

The problem. Generate a circuit, using no more than 10 qubits, that approximates the unitary e^{-iH} where H

is the qubit Hamiltonian of a **LiH** (lithium hydride) molecule. The **LiH** Hamiltonian is composed of 276 Pauli strings, and can be found HERE. The approximation error is defined in the next section, and should be less than 0.1. The circuit should be composed of the CX and single qubit gates only.

Simulation in nutshell. Let's begin by introducing notation, adhering to standard definitions; a quantum state $|\psi\rangle$ is a vector in a Hilbert space, and a Hamiltonian is a Hermitian operator acting on that space. As physical fundamentals are not the main focus of the paper, we will merely mention, without providing justification, that a quantum state at time t in the presence of a Hamiltonian H is given by the relation $|\psi(t)\rangle = e^{-iHt} |\psi(0)\rangle$. Note that, due to the fact that H is a Hermitian operator, e^{-iHt} is a unitary operator.

For example, suppose that $|\psi\rangle = |00\rangle$ and $H = Z \otimes Z$. Thus, the time evolution operator is $e^{itZ\otimes Z}$. As $|00\rangle$ is an eigenstate of H, since $Z \otimes Z |00\rangle = |00\rangle$, we have that $|\psi(t)\rangle = e^{-it}|00\rangle$. However, if $|\psi\rangle = |00\rangle + |10\rangle$ (ignoring normalization factors), then we would have that $|\psi(t)\rangle = e^{-it}|00\rangle + e^{it}|10\rangle$.

For simulating a general Hamiltonian, recall that the Pauli matrices, along with the identity, span the space of matrices. Thus, one can assume that the Hamiltonian is given as a linear combination of Paulis. We will denote it as $H = \sum_i H_i$, where each of the H_i is a Pauli product multiplied by a scalar factor.

Using the Lie-Trotter formula [Tro59]

$$e^{A+B} = \lim_{n \to \infty} (e^{A/n} e^{B/n})^n$$

for arbitrary matrices A, B, we obtain that one can apply the gates $e^{\Delta t H_i}$ for $1/(\Delta t)$ iterations and expect to converge to the desired product. This insight defines a circuit which we will refer to as the naive approach.

Naive approach. Before diving into the technical details, let's first review our competitor. Consider the straightforward assignment Lie-Trotter [Tro59], which handles each of the Hamiltonian terms one by one. Assume that the Hamming weight of the support of H_i is equal to d_i . Thus, we need two steps to rotate each wire into the parity base (relative to the Hamiltonian) and uncompute it in the end. Then, we can apply the CX from each qubit in the support to a (arbitrary) chosen

one, which will sum the wires' parity and finally propagate the sum through the $RZ\left(\theta\Delta t\right)$ gate (rotation by the coefficient of the term and the step size). Therefore, in total we pay $2+2d_i$ for each term and the whole circuit will require:

$$D^{\text{naive}}(n,m) = \sum_{i} 2 + 2d_i = m \sum_{i} \frac{2 + 2d_i}{m} = 2m \left(1 + \mathbb{E}_{\sim i}[H]\right)$$

Where $D^{\mathrm{naive}}\left(n,m\right)$ is the depth of the circuit which computes a single Δt step of the Lie-Trotter formula[Tro59]. To gain a better understanding of what constitutes a good solution, let us assume for the moment that $\mathbb{E}_{\sim i}[H] > 5$. Under this assumption, the naive approach yields a circuit of depth $\geq 2 \cdot 276 \cdot 6 \sim 3100$. Ultimately, we will see that our solution produces a 1759-depth circuit.

Characterize our LiH. A natural question to ask is whether there exists a more efficient way to present the Hamiltonian. Although we cannot be certain that there is not, let's try to estimate how much improvement can be achieved by applying the Bravyi-Kitaev transformation [BK02]. In one sentence, Bravyi and Kitaev have shown that the ladder and annihilation operators have Pauli representations such that each operator touches at least $\log(n)$ of the qubits. Since the Hamiltonian of molecule¹ in the energy base has a "product" of four operators (i.e $a_i^{\dagger} a_i^{\dagger} a_n a_m$), we get that there may be operators in the computing base which touch $4 \log (n)$ qubits. In our case, $n = 10 \Rightarrow 4\log(n) > n$. In conclusion, it seems that the known techniques for reducing the support of the terms asymptotically do not work here. This is an indication that a 10-fold improvement is unlikely. Therefore, we will aim for a 2-fold improvement over the naive approach.

2 Single Term Heuristics.

Main wire principle. From now on, we will refer to the wire that sums the parity of the state as the main wire. For a given term, we have chosen the main wire to be the median of the wires in its support. For example, consider the following term:

XXXXIII

 $^{^1{\}rm Sometimes}$ called electronic structure Hamiltonian

Then, as the second wire is above (and beneath) half of the wires, it will be the main wire in that case. The separation into upper and lower sides will enable us to guarantee that at least two CX will be added in parallel (using the next heuristic).

Upper and lower summations. We sum the parity of the given term as follows. Denote by $i_1, i_2, ... i_W, ... i_k \subset [n]$ the indices in the support of the

given term, such that i_W is the main wire. After rotating the wires into the phase base, we sum the parity of $i_1, i_2, ... i_{W-1}$ into i_{W-1} by applying a $CX(i_j, i_{W-1})$ for j < W - 1.

Since all the segments $[i_j, i_{W-1}]$ are disjoint from any segments of the form $[i_{W+1}, i_j]$ for j > W+1, we can sum the parity of $i_{W+1}, i_{W+2}..i_k$ wires into i_{W+1} in parallel. This heuristic reduces the depth of a single term by almost half.



Figure 1: Demonstrating the above methods applied to the terms XIXZZXIXXZ and XIXXZXIXXZ, the fifth wire (qubit) is the main wire that sums the parity.

3 Greedy Order Heuristics.

Next, we will review our methods to impose the gates order. Clearly the order does matter, to see that consider an hamiltonian pair which differ by only single coordinate i.e XXXX and XXXZ. In that case the contribute of the first tree qubits remain the same for both of them, and therefore there is no needed to uncompute

them after the applying of the first gate.

Sample Greedy Diameters. Assume that we given a set of therms with a promise that it might has a lot of "closer" terms (relative to the Hamming distance of their supports). Our first simple heuristic goal is to chain them together in greedy manner such that the distance of each adjacent terms will be minimal.

Algorithm 1: Chain an Hamiltonian set

```
while G \neq \mathbf{do}
T \leftarrow \text{Greedy Spanning Tree } (G)
D_i \leftarrow \text{Diameter } (T)
G \leftarrow G/Di
\mathbf{return } D_0...D_l
```

It's worth to note that replacing the Greedy Spanning Tree by MST (and the diameter by just the DFS scanning order of the MST) could be proving as 2-OPT in the terms of chaining. We think about the uncompute stage as the climbing back at the tree, and therefore $\mathbf{OPT} \leq 2w$ (MST). But also it easy to see that every path which contain all the vertices has weight greater than w (MST), Hence 2w (MST) $\leq 2\mathbf{OPT}$. Yet, By the fact that we have already improved the naive solution at factor which close to $\frac{1}{2}$ and by the intuition that the optimal depth is not very far from the naive (don't expect

Technically we have separated those groups by follow, first we have sorted the term collection by the follow weight functions f, g such that:

$$f(H_i) = \min j : H_{j'} = I \text{ for } j' > j$$

 $g(H_i) = \max j : H_{j'} = I \text{ for } j' \le j$

And then we have matched pairs till we get into intersection.

that $D^{\text{naive}} \geq 4\text{OPT}$ we decided to stick to the greedy construction, which in the next chapter will generalized to the product graph.

Hyperplanes separation. The cost of chaining pair of Hamiltonians isn't a monotone function of their Hamming distance, for example consider the case of two terms which have not overlapping at all, i.e *XXXXIIII* and *IIIIXXXX* then it's clear that those two can be impose such they will be computed in parallel.

4 The Product Graph.

The main disadvantage of the last separation is that it doesn't take into a count any relation between terms which classified in the same side of the plane. The suggested solution: We will say that the pair H_i, H_j share a "solid product" if $f(H_i) \leq g(H_j)$ where f, g defined above. We denote such relation by $(H_i, H_j) \in \mathbf{SP}$. Define the graph $G^2 = (V \times V, E')$ where V is the set of the terms, to be:

$$E' = \{\{(u, v), (w, z)\} : (u, v), (w, z), (u, z), (w, v) \in \mathbf{SP}\}$$

$$w(e) = w(\{(u, v), (w, z)\}) = \max w(u, w), w(v, z)$$

Note, that in the G^2 path which pass through all the vertices contain an |V| copies of each term. Hence looking for spanning trees makes no sense. Therefore our diameters sampler will be more appropriate for that task (when we ensure that each sampled tree contains at most one copy of each term).

Another advantage of this method that it can be easily generalize to scanning triples or higher orders by taking higher power of the graph, the main disadvantage is (of course) processing time.

The final submission is a mixture of all the methods we have reviewed above.

References

[Tro59] H. F. Trotter. "On the Product of Semi-Groups of Operators". In: *Proceedings of the American Mathematical Society* 10.4 (1959), pp. 545–551. ISSN: 00029939, 10886826. URL: http://

www.jstor.org/stable/2033649 (visited on 07/28/2023).

[BK02] Sergey B. Bravyi and Alexei Yu. Kitaev. "Fermionic Quantum Computation". In: Annals of Physics 298.1 (May 2002), pp. 210–226. DOI: 10.1006/aphy.2002.6254. URL: https://doi.org/10.1006%2Faphy.2002.6254.

```
1
2 from importlib.resources import path
3 from math import ceil
4 from struct import unpack
5 from qiskit import QuantumCircuit
6 from qiskit.visualization import circuit_drawer
7 from matplotlib import pyplot as plt
8 import datetime
9 import numpy as np
10
                            = 4
11 N
12 WIRES
                            = 10
                            = "NOT"
13 TEST
14 RANDOMSTACTICS
                            = True
                            = "PRODUCT"
15 STRATEGY
16 OUTSIDE_OF_CIRCUIT
                            = WIRES + 1
17 PATHFILE
                                "NOT" : "./LiH",
18
                                "YES" : "./LiH_test",
19
                                "PRODUCT" : "./LiH_test_product_space"
20
21
                                } [TEST]
22
23
   class local_Hamiltonian():
24
       def __init__(self, tensor, weight) -> None:
25
           self.tensor = tensor
26
27
           self.weight = weight
28
           self.parent = self
29
30
       def tensorspace(self, other) -> bool:
31
           for A,B in zip( list(self.tensor), list(other.tensor)):
               if "I" not in [A, B]:
32
33
                   return False
34
           return True
35
       def dis(self, other) -> int:
36
37
           for A,B in zip(list(self.tensor), list(other.tensor)):
38
39
               if A != B:
40
                   ret += 1
41
           return ret
42
43
       def solid_product(self, other):
44
           indices = []
45
           for j in range(1,WIRES):
46
               l= "X" * j + "I" * (WIRES-j)
               r = "I" * j + "X" * (WIRES-j)
47
48
               if local_Hamiltonian(1,0).tensorspace(self) and\
49
                    local_Hamiltonian(r,0).tensorspace(other):
50
                   indices.append(j)
51
52
           if len(indices) > 0:
53
               return True, indices
54
           return False, []
55
56
       def newbase(self, perm):
57
           tensor = [ "" ] * len(self.tensor)
58
           for i in range(len(perm)):
59
               tensor[i] = self.tensor[perm[i]]
60
           ret = local_Hamiltonian(tensor,self.weight)
61
           ret.parent = self.parent
62
           return ret
63
```

```
64
        def median(self):
 65
            support = list(filter( lambda x : self.tensor[x] != 'I', range(WIRES)))
 66
            if len(support) != 0:
 67
                return support[int(len(support)/2)]
 68
 69
 70
        def seconed_wires(self):
 71
            j = self.median()
 72
 73
            def find_nearset_nontrival(sign):
 74
 75
                gen = [ (i,v) for i,v in list(enumerate(self.tensor))[j+1:]] if sign ==1\
 76
 77
                     else [ (i,v) for i,v in list(enumerate(self.tensor))[j-1::-1]]
 78
 79
                for i,v in gen:
 80
                    if v != "I":
 81
                         return i
 82
                return j
 83
            pos_wire = j
 84
 85
            neg_wire = j
            if j + 1 < WIRES:
 86
 87
                pos_wire = find_nearset_nontrival(1)
            if j - 1 >= 0:
 88
 89
                neg_wire = find_nearset_nontrival(-1)
 90
 91
            return neg_wire, pos_wire
 92
 93 def parser_line(line) -> local_Hamiltonian:
 94
        line = line.split()
 95
        return local_Hamiltonian( list(line[-1]),
         { "-" : -1 , "+" : 1 }[line[0]] * np.float64(line[1]) / N )
 96
 97
 98 def parser() -> None:
 99
        hamiltonians = [ ]
100
        for line in open(PATHFILE).readlines():
101
            if len(line) > 1:
102
                hamiltonians.append(parser_line(line))
103
        return hamiltonians
104
105 def donothing(_):
106
        pass
107
108 def rotateY(cir):
109
        def _func(wire):
110
            cir.s(wire)
111
            cir.h(wire)
112
        return _func
113
114 def unrotateY(cir):
115
        def _func(wire):
116
            cir.h(wire)
117
            cir.sdg(wire)
118
        return _func
119
120 def MulByterm (circuit : QuantumCircuit, term ,next_terms = [], last_terms = [],
     main_wire = WIRES-1) -> QuantumCircuit:
121
122
123
        def reqursive_manner(tensor, wire, weight, last_wire, _sign, first_not_trival=True):
124
125
            if wire < 0 or wire == WIRES:</pre>
126
                return QuantumCircuit(WIRES),QuantumCircuit(WIRES)
127
```

```
128
            compute =
                       {
129
                "X" : lambda cir : cir.h,
                "Y" : lambda cir : rotateY(cir),
130
                "Z" : lambda cir : donothing,
131
                "I" : lambda cir : donothing }
132
133
134
135
            uncompute = {
136
                "X" : lambda cir : cir.h,
                "Y" : lambda cir : unrotateY(cir),
137
                "Z" : lambda cir : donothing,
138
                "I" : lambda cir : donothing }
139
140
141
            pauli = tensor[wire]
142
            if wire == main_wire:
                circuit_node = QuantumCircuit(WIRES)
143
                LU, RU = reqursive_manner(tensor, wire-1, weight, last_wire, -1, first_not_trival = True)
144
                LD, RD = reqursive_manner(tensor, wire+1, weight, last_wire, 1, first_not_trival = True)
145
146
                compute[pauli](circuit_node)(wire)
147
148
                for L in [ LD, LU]:
149
                    circuit_node = circuit_node.compose(L)
150
                circuit_node.rz(2*weight, main_wire)
151
                for R in [ RD, RU]:
152
                    circuit_node = circuit_node.compose(R)
153
                uncompute[pauli](circuit_node)(wire)
154
                return circuit_node
155
            if pauli == 'I':
156
157
158
159
                # temp_wire = last_wire + _sign if first_not_trival else last_wire
160
                return reqursive_manner(tensor, wire + _sign, weight,
161
                 last_wire, _sign, first_not_trival = first_not_trival)
162
            else:
163
164
165
                temp_wire = wire if first_not_trival else last_wire
166
                parity_collector = False
167
                if first_not_trival:
168
                    parity_collector = True
169
170
171
                L, R = reqursive_manner(tensor, wire + _sign, weight, temp_wire, _sign, first_not_trival =
False)
172
                circuit_left, circuit_right = QuantumCircuit(WIRES),QuantumCircuit(WIRES)
173
174
                if (parity_collector) or not (\
                     (last_terms[wire][0] == pauli) and\
175
176
                      (last_terms[wire][1][ { 1 : 1 , -1 : 0 }[_sign] ] == last_wire)):
177
178
                    compute[pauli](circuit_left)(wire)
179
                    circuit_left = circuit_left.compose(L)
180
                    circuit_left.cx(wire, last_wire)
181
182
                else:
183
                    circuit_left = L
184
185
                if (parity_collector) or not (
186
                     (next_terms[wire][0] == pauli) and\
187
                      (next_terms[wire][1][ { 1 : 1 , -1 : 0 }[_sign] ] == last_wire)):
188
189
                    circuit_right.cx(wire, last_wire)
190
                    circuit_right = circuit_right.compose(R)
```

```
191
192
                    uncompute[pauli](circuit_right)(wire)
193
                else:
194
                    circuit_right = R
195
                return circuit_left, circuit_right
196
197
        circuit = circuit.compose(reqursive_manner(term.tensor, main_wire, term.weight, main_wire, 0))
198
        return circuit
199
200
201 def cutting(circuit : QuantumCircuit):
        '''Second optimization, cuts the gates which are follwed by their
202
203
        uncompute '''
204
        def filter_by_wire(wire):
205
            return list(filter( lambda item :\
206
                any( [register.index == wire for register in item[1][1]] ), enumerate(circuit.data) ))
207
208
        UNCOMPTE = {
209
            "h" : "h",
210
            "cx" : "?",
211
            "rz" : "?"
212
            "s" : "sdg",
213
            "sdg" : "s",
214
            "sxdg" :"?" }
215
216
217
        indices_todelete = []
218
        for wire in range(WIRES):
219
            operators = filter_by_wire(wire)
220
            j = 0
221
            while (j < len(operators) - 1 ):</pre>
222
                if ( UNCOMPTE[operators[j][1][0].name] == operators[j+1][1][0].name ):
223
                    indices\_todelete.append( \ operators[j][\emptyset] \ )
224
225
                    indices_todelete.append( operators[j+1][0] )
                j += 1
226
227
228
        for index in reversed(sorted(indices_todelete)):
229
            circuit.data.pop(index)
230
231
        return circuit
232
233
234 def genreate_circut(terms = None):
235
        circuit = QuantumCircuit(WIRES)
236
        terms = parser() if terms == None else terms
237
        print(len(terms))
238
        for i, term in enumerate(terms):
239
240
            main_wire = term.median()
241
            next_terms,last_terms = [],[]
242
            for _j in range(WIRES):
243
244
                found = False
245
                if i+1 < len(terms):</pre>
246
                     for _term in terms[i+1:]:
247
                         if _term.tensor[_j] != 'I':
248
                             next_terms.append( (_term.tensor[_j], _term.seconed_wires() ))
249
                             found = True
250
                             break
251
                if not found:
                    next_terms.append( ('I', OUTSIDE_OF_CIRCUIT ))
252
253
254
                found = False
```

```
if i > 0:
255
256
                    for _term in terms[i-1::-1]:
257
                        if _term.tensor[_j] != 'I':
258
                            last_terms.append( (_term.tensor[_j], _term.seconed_wires() ))
                            found = True
259
                            break
260
                if not found:
261
                    last_terms.append( ('I', OUTSIDE_OF_CIRCUIT ))
262
263
264
            circuit = MulByterm(circuit, term, main_wire=main_wire,
265
             next_terms=next_terms, last_terms=last_terms )
266
267
        return circuit
268
269 def genreate_optimzed_circut(circuit, terms, svg =False, entire = False):
        circuit = cutting(cutting(circuit))
270
271
        if entire:
272
273
            for _ in range(ceil(np.log(N))):
274
                circuit = circuit.compose(circuit)
275
            circuit = cutting(circuit)
276
        print(f"TERMS: {len(terms)}, DEPTH:{circuit.depth()}")
277
278
279
        if svg:
280
           circuit_drawer(circuit, output='mpl',style="bw", fold=-1)
281
            plt.title( f"TERMS: {len(terms)}, DEPTH:{circuit.depth()}")
282
            plt.tight_layout()
283
            plt.savefig(f'Ham_{STRATEGY}-{datetime.datetime.now()}.svg')
284
285
        if entire:
286
            open(f"Ham_{STRATEGY}-{datetime.datetime.now()}.qasm", "w+").write(circuit.qasm())
287
288
        return circuit.depth()
289
290
```

291

```
1 from copy import deepcopy
 2 import networkx as nx
3 from Hamiltonian_parser import WIRES, parser, local_Hamiltonian,genreate_circut,genreate_optimzed_circut
 4 from itertools import permutations, product, combinations
 5 from random import choice
6 import pickle as pkl
8
  from matplotlib import pyplot as plt
9
10
11 class Permutation_Base():
12
       def __init__(self, arr) -> None:
           self.arr = arr
13
           self.parent = self
14
15
16
17 def generated_the_product_graph(terms = parser()):
18
       def generated_the_product_graph_by_base(_terms, number_premu=0):
19
20
                    = nx.Graph()
           Gproduct = nx.Graph()
21
22
23
           edges_set = set()
24
25
           for (H1, H2) in product(_terms, _terms):
26
               if H1.tensorspace(H2):
27
                   G.add_edge(H1, H2)
28
                   edges_set.add( (H1,H2) )
29
                   G.edges[(H1, H2)]['weight'] = H1.dis(H2)
30
                   G.edges[(H1, H2)]['solid'] = True
31
                   G.edges[(H1, H2)]["permutation"] = j
32
           for e in G.edges():
33
34
               H1,H2 = e
35
               for H3,H4 in \
36
                   product(list(G.adj[H1]), list(G.adj[H2])):
37
                   if (H1,H4) in edges_set and\
38
                        (H3,H4) in edges_set and
39
                            (H2,H3) in edges_set:
40
                       Gproduct.add_edge((H1,H2), (H3, H4))
41
                       Gproduct.edges[(H1, H2), (H3, H4)]['weight'] = max(H1.dis(H3), H2.dis(H4))
                       Gproduct.edges[(H1, H2), (H3, H4)]['solid'] = True
42
43
                       Gproduct.edges[(H1, H2), (H3, H4)]["permutation"] = j
44
45
           for (H1, H2) in product(_terms, _terms):
46
               if (H1,H2) not in edges_set:
47
                   G.add_edge(H1, H2)
48
                   G.edges[(H1, H2)]['weight'] = H1.dis(H2)
49
                   G.edges[(H1, H2)]['solid'] = False
50
                   G.edges[(H1, H2)]["permutation"] = j
51
52
           print(f"vertices:{Gproduct.number_of_nodes()}\t edges: ~{Gproduct.number_of_edges()}")
53
           return Gproduct, G, _terms
54
55
       return pkl.load( open(f"mainG-276-1.pkl", "br"))
56
57
       permutations = list(map(lambda x: Permutation_Base(x) , [
58
           # [0,2,4,6,8,1,3,5,7,9],
59
           [0,1,2,3,4,5,6,7,8,9]
60
           # [0,7,4,6,8,9,3,5,2,1]
61
       ]))
62
       graphs = []
63
       perm_terms = []
```

```
64
        mainG = nx.Graph()
65
        mainProductG = nx.Graph()
 66
        for j, permutation in enumerate(permutations):
 67
            perm\_terms = list(map( lambda x : x.newbase(permutation.arr), terms))
 68
            productG, G, _ = generated_the_product_graph_by_base(perm_terms, number_premu=j)
 69
 70
            if mainG.number_of_nodes() == 0:
 71
                mainG = nx.compose(mainG, G)
 72
            mainProductG = nx.compose(mainProductG, productG)
73
74
        pkl.dump((mainG, mainProductG, terms ,permutations), open(f"mainG-{len(terms)}-
{len(permutations)}.pkl", "bw+"))
75
        return mainG, mainProductG, terms, permutations
76
 77 def select(_list, v, G, flag = True):
        minimal = min(_list, key =lambda u : G.edges[v,u]['weight'] )
 78
 79
        return choice( [r for r in _list if G.edges[v,r]['weight'] == \
80
             G.edges[v,minimal]['weight'] ])
81
82 def notcolorized(node, _set):
        if isinstance(node, tuple):
83
            term1, term2 = node
84
85
            if (term1.parent in _set) or (term2.parent in _set):
86
                return False
 87
            return True
 88
        else:
 89
            return node.parent not in _set
90
91 def colirize(node, _set):
92
        if isinstance(node, tuple):
93
            term1, term2 = node
94
95
            if (term1.parent in _set) or (term2.parent in _set):
96
                return False, _set, 1
97
            _set.add(term1.parent)
98
            _set.add(term2.parent)
99
            return True
100
        else:
101
            if node.parent in _set:
102
                return False
103
            else:
104
                _set.add(node.parent)
105
                return True
106
107
108 #randomized DFS.
109 def sample_path(G, terms) -> tuple((nx.Graph, set)):
110
        print("sample_path")
111
        color = set()
112
113
        def DFS(v, T, _color, sign=0, flag =True):
114
115
            if not colirize(v, _color):
116
                return _color, sign
117
            can_packed = [None]
118
            while len(can_packed) > 0 :
119
120
                can_packed = list(filter(lambda x :\
121
                    notcolorized(x, _color), G.adj[v]))
122
123
                if len(can_packed) > 0:
124
                    u = select( can_packed, v, G, flag=flag)
125
                    T.add_edge(v,u)
126
                    if 'permutation' in G.edges[v,u]:
```

```
127
                        T.edges[v,u]['permutation'] = G.edges[v,u]['permutation']
128
                    T.edges[v,u]['sign'] = sign
129
                    _color, sign = DFS(u, T, _color, sign=sign+1, flag=flag)
130
            return _color, sign
131
132
        T = nx.DiGraph()
133
        1 = []
134
135
        for v in G.nodes():
136
            T.add_node(v)
137
            1.append(v)
138
139
        _sign = 0
140
        for v in G.nodes():
            color, _sign = DFS(v, T, color, sign=_sign, flag=False)
141
142
143
        return T, color
144
145 def get_Diameter(Tree: nx.Graph) -> tuple((tuple(([], int))), tuple(([], int)))) :
146
147
        def DFS_tree_depth(G, v):
148
149
            if len(list(G.adj[v])) == 0 :
150
                return (([v],1),([v],1))
151
            branches = []
152
            maxinnerpath, maxinnerdepth = [],0
153
            for u in list(G.adj[v]):
154
155
                ((temppath, tempdepth), \
156
                    (tempinnerpath, tempinnerdepth)) = DFS_tree_depth(G,u)
157
158
                if maxinnerdepth < tempinnerdepth:</pre>
159
                    maxinnerpath, maxinnerdepth = tempinnerpath, tempinnerdepth
160
161
                branches.append((temppath, tempdepth))
162
            maxpath, maxdepth = [],0
163
164
165
            for (b1, d1), (b2, d2) in combinations(branches, r=2):
166
                if 1 + d1 + d2 > maxinnerdepth:
167
                    maxinnerpath
                                    = b1 + [v] + b2
168
                    maxinnerdepth
                                   = 1 + d1 + d2
169
170
            for b,d in branches:
171
                if 1 + d > maxdepth:
172
                    maxpath = [v] + b
173
                    maxdepth = 1 + d
174
            return ((maxpath,maxdepth), (maxinnerpath, maxinnerdepth))
175
176
        ((maxpath, maxdepth), (maxinnerpath, maxinnerdepth)) = DFS_tree_depth(Tree, list(Tree.nodes)[0])
177
        print(maxdepth, maxinnerdepth)
178
        return maxpath if maxdepth > maxinnerdepth else maxinnerpath
179
180
181 def generate_simple_graph(_terms):
182
        G = nx.Graph()
183
        for (H1, H2) in product(_terms, _terms):
184
            G.add_edge(H1, H2)
185
            G.edges[(H1, H2)]['weight'] = H1.dis(H2)
186
        return G
187
188 def greedy_path( terms ):
189
        G = generate_simple_graph(terms )
190
```

```
191
        def reqursive_form( _terms ):
192
            if len(_terms) < 3:</pre>
193
                return _terms
194
            else:
195
196
                T, _ = sample_path(G, _terms)
197
198
                Q = get_Diameter(T)
199
                color = set()
200
201
                ret = []
                for H in Q:
202
203
                    if H.parent not in color:
204
                         color.add(H.parent)
205
                         ret.append(H.parent)
206
                         G.remove_node(H)
207
208
                remain_terms = [ term for term in _terms if term not in color ]
209
                return ret + reqursive_form(remain_terms)
210
        return reqursive_form(terms)
211
212 def Hamiltonian_sorting(hamiltonians):
        groups = [[] for _ in product(range(WIRES), range(WIRES))]
213
214
        for term in hamiltonians:
215
            x,y = term.seconed_wires()
            groups[x + WIRES*y].append(term)
216
217
218
        ret = [ ]
219
        for group in groups:
220
            group = greedy_path(group)
221
            ret += group
222
        return ret
223
224 def enforce_seapration(hamiltonians):
225
        def seperate(terms):
226
227
228
            def sort_tensor_by_geometrical_support(tensor, up = True):
229
230
231
                for j,pauli in enumerate( { True: tensor, False: reversed(tensor) }[up] ):
232
                     if pauli != "I":
233
                         if up:
234
                             return 10 - j
235
                         else:
236
                             return 10 - j
237
                else:
238
                    return 10
239
240
                   = sorted(terms,\
241
                 key=lambda x : sort_tensor_by_geometrical_support(x.tensor, up=True))
242
            beneath = sorted(terms, \
243
                 key=lambda x : sort_tensor_by_geometrical_support(x.tensor, up=False))
244
245
            contact_point = min( range(len(terms)), \
246
                key = lambda i : above[i].tensorspace(beneath[i]))
247
248
            print(f"contact point: {contact_point}")
249
            path = []
250
            for x,y in zip( sorted(above[:contact_point], key = lambda z : z.tensor),\
251
                sorted(beneath[:contact_point], key = lambda z : z.tensor)):
252
                path.append(x)
253
                path.append(y)
254
```

```
255
            new\_terms = []
256
            for x in above[contact_point:]:
257
                 if x not in beneath[:contact_point]:
258
                     new_terms.append(x)
259
            return new_terms, path
260
        path = [ ]
261
        terms ,temppath = seperate(deepcopy(hamiltonians))
262
        path += temppath
263
        print(f"terms:{len(terms)}")
264
265
        return path, terms
266
267 def alternate_path_v2(mG : nx.Graph, G : nx.Graph, \
         terms, permutations, single_iteration = False):
268
269
        other_color = set()
270
271
        T, _ = sample_path(G, terms)
272
        Q = get_Diameter(T)
273
        ret = []
274
275
        for (u,v) in Q:
            for H in [u,v]:
276
                if H not in other_color:
277
                    other_color.add(H)
278
279
                     ret.append(H)
280
281
        made_progress = True
282
        _single_iteration = True
283
        while _{\rm single\_iteration} and (_{\rm len}(Q) > 2 and made_{\rm progress}):
284
            made_progress = False
285
            for u,v in Q:
286
                for H in [u,v]:
287
                     if H.parent not in other_color:
288
                         ret.append(H)
289
                         other_color.add(H.parent)
290
                     for w in list(G.nodes()):
291
                         if H in w:
292
                             G.remove_node(w)
293
                             made_progress = True
294
295
                     if mG.has_node(H):
296
                         mG.remove_node(H)
297
298
            T, _ = sample_path(G, terms)
299
            if T.number_of_edges() > 1:
300
                Q = get_Diameter(T)
301
            else:
302
303
            _single_iteration = not single_iteration
304
305
        if not single_iteration:
306
            for term in terms:
307
                 if term.parent not in other_color:
308
                     ret.append(term)
309
310
        return ret, terms, other_color
311
312 def main_enforce(hamiltonians):
313
        path, terms = enforce_seapration(hamiltonians)
314
        path += Hamiltonian_sorting(terms)
315
        circuit = genreate_circut(path)
        depth = genreate_optimzed_circut(circuit, hamiltonians, svg=False, entire=False)
316
317
        return circuit
318
```

```
319 def compose_alternate_enforce():
320
            G, mainProductG, terms, permus = generated_the_product_graph()
321
            canidates = [ ]
322
            for _ in range(5):
323
                path, terms, color = alternate_path_v2( deepcopy(G), deepcopy(mainProductG), terms, permus,
single_iteration=True)
324
                circuit = genreate_circut(path)
325
                genreate_optimzed_circut(circuit, terms, svg=False, entire=False)
                remain_terms = [ term for term in terms if term not in color ]
326
327
                circuit = circuit.compose( main_enforce( remain_terms ) )
                depth = genreate_optimzed_circut(circuit, terms, svg = False, entire=False )
328
329
                canidates.append( (depth, circuit) )
                print(f"DEPTH: {depth}")
330
331
            depth, circuit = min(canidates, key = lambda x : x[0])
332
            depth = genreate_optimzed_circut(circuit, terms, svg = False, entire=True)
333
            # depth = genreate_optimzed_circut(circuit, terms, entire=True)
334
            return circuit,terms
335
336
337
338
339 def demonstrate_fig( ):
        path = [ local_Hamiltonian( "XIXZZIIIII", 0.5 ),
340
341
          local_Hamiltonian( "XXXZIIIIIII", 0.5 ),
          local_Hamiltonian( "IIIIIIXXZX", 0.5 ),
342
343
          local_Hamiltonian( "IIIIIIIZIZX", 0.5 ),
344
          local_Hamiltonian( "IIIXIIZIZX", 0.5 ) ]
345
346
        path, terms = enforce_seapration(path)
347
        circuit = genreate_circut(path)
348
        genreate\_optimzed\_circut(circuit ,path, \ svg= \colored{True}, \ entire= \colored{False})
349
350
351 if __name__ == "__main__":
352
353
        circuit, terms = compose_alternate_enforce()
354
355
356
```

```
1 + 0.003034656830204855 * IIIYYIIIYY
 2 + 0.003034656830204855 * IIIXXIIIYY
 3 + 0.003034656830204855 * IIIYYIIIXX
                          * IIIXXIIIXX
 4 + 0.003034656830204855
                          * YZZZYIIIYY
 5 - 0.008373361424264817
                          * XZZZXIIIYY
6 - 0.008373361424264817
                          * YZZZYIIIXX
7 - 0.008373361424264817
                         * XZZZXIIIXX
8 - 0.008373361424264817
                           * YZZYIIIIYY
9 + 0.00211113766859809
                           * XZZXIIIIYY
10 + 0.00211113766859809
                           * YZZYIIIIXX
11 + 0.00211113766859809
                           * XZZXIIIIXX
12 + 0.00211113766859809
                           * IIIIIIIYY
13 - 0.00491756976241806
                           * IIIIIIIXX
14 - 0.00491756976241806
15 + 0.010540187409026488 * ZIIIIIIIIYY
16 + 0.010540187409026488 * ZIIIIIIIIXX
17 - 0.0011822832324725804 * IZIIIIIIIYY
18 - 0.0011822832324725804 * IZIIIIIIIXX
19 - 0.0011822832324725804 * IIZIIIIIYY
20 - 0.0011822832324725804 * IIZIIIIIXX
                         * IIIZIIIIYY
21 - 0.00154067008970742
                           * IIIZIIIIXX
22 - 0.00154067008970742
23 + 0.011733623912074194 * IIIIZIIIYY
24 + 0.011733623912074194 * IIIIZIIIXX
25 + 0.0027757462269049522 * IIIIIIZIIYY
26 + 0.0027757462269049522 * IIIIIIZIIXX
27 + 0.0036202487558837123 * IIIIIIIZIYY
28 + 0.0036202487558837123 * IIIIIIIZIXX
29 + 0.0036202487558837123 * IIIIIIIIZYY
30 + 0.0036202487558837123 * IIIIIIIIZXX
31 + 0.005996760849734561 * IIYZYIIYZY
32 + 0.005996760849734561 * IIXZXIIYZY
33 + 0.005996760849734561 * IIYZYIIXZX
34 + 0.005996760849734561
                          * IIXZXIIXZX
                          * IIYYIIIYZY
35 + 0.004802531988356293
                          * IIXXIIIYZY
36 + 0.004802531988356293
                          * IIYYIIIXZX
37 + 0.004802531988356293
                          * IIXXIIIXZX
38 + 0.004802531988356293
                          * YZYIIIIYZY
39 - 0.004879740484191497
                          * XZXIIIIYZY
40 - 0.004879740484191497
                          * YZYIIIIXZX
41 - 0.004879740484191497
                          * XZXIIIIXZX
42 - 0.004879740484191497
                          * IYZZYIYZZY
43 + 0.005996760849734561
                          * IXZZXIYZZY
44 + 0.005996760849734561
45 + 0.005996760849734561
                          * IYZZYIXZZX
46 + 0.005996760849734561
                          * IXZZXIXZZX
47 + 0.004802531988356293
                          * IYZYIIYZZY
48 + 0.004802531988356293
                          * IXZXIIYZZY
49 + 0.004802531988356293
                          * IYZYIIXZZX
50 + 0.004802531988356293
                          * IXZXIIXZZX
51 - 0.004879740484191497
                          * YYIIIIYZZY
                          * XXIIIIYZZY
52 - 0.004879740484191497
                          * YYIIIIXZZX
53 - 0.004879740484191497
                          * XXIIIIXZZX
54 - 0.004879740484191497
                          * IIIYYYZZZY
55 - 0.008373361424264817
                          * IIIXXYZZZY
56 - 0.008373361424264817
57 - 0.008373361424264817 * IIIYYXZZZX
58 - 0.008373361424264817 * IIIXXXZZZX
                           * YZZZYYZZZY
59 + 0.0307383271773138
                           * XZZZXYZZZY
60 + 0.0307383271773138
                          * YZZZYXZZZX
61 + 0.0307383271773138
                        * XZZZXXZZZX
62 + 0.0307383271773138
63 - 0.0077644411821215335 * YZZYIYZZZY
```

```
64 - 0.0077644411821215335 * XZZXIYZZZY
 65 - 0.0077644411821215335 * YZZYIXZZZX
 66 - 0.0077644411821215335 * XZZXIXZZZX
 67 - 0.005949019975734247 * IIIIIYZZZY
 68 - 0.005949019975734247 * IIIIIXZZZX
                         * ZIIIIYZZZY
 69 - 0.0351167704024114
                         * ZIIIIXZZZX
 70 - 0.0351167704024114
 71 + 0.0027298828353264134 * IZIIIYZZZY
 72 + 0.0027298828353264134 * IZIIIXZZZX
 73 + 0.0027298828353264134 * IIZIIYZZZY
 74 + 0.0027298828353264134 * IIZIIXZZZX
 75 + 0.0023679368995844726 * IIIZIYZZZY
 76 + 0.0023679368995844726 * IIIZIXZZZX
 77 - 0.03305872858775587 * IIIIZYZZZY
 78 - 0.03305872858775587 * IIIIZXZZZX
 79 - 0.0021498576488650843 * IIIIIYIZZY
 80 - 0.0021498576488650843 * IIIIIXIZZX
 81 - 0.0021498576488650843 * IIIIIYZIZY
 82 - 0.0021498576488650843 * IIIIIXZIZX
 83 + 0.004479074568182561 * IIIIIYZZIY
 84 + 0.004479074568182561 * IIIIIXZZIX
 85 + 0.004802531988356293 * IIYZYIIYYI
 86 + 0.004802531988356293 * IIXZXIIYYI
 87 + 0.004802531988356293 * IIYZYIIXXI
 88 + 0.004802531988356293
                          * IIXZXIIXXI
 89 + 0.010328819322301622
                           * IIYYIIIYYI
 90 + 0.010328819322301622
                           * TTXXTTTYYT
 91 + 0.010328819322301622
                           * IIYYIIIXXI
 92 + 0.010328819322301622
                           * IIXXIIIXXI
93 - 0.003466391848475337
                           * YZYIIIIYYI
94 - 0.003466391848475337
                           * XZXIIIIYYI
95 - 0.003466391848475337
                           * YZYIIIIXXI
96 - 0.003466391848475337
                           * XZXIIIIXXI
97 + 0.004802531988356293
                           * IYZZYIYZYI
98 + 0.004802531988356293
                           * IXZZXIYZYI
99 + 0.004802531988356293
                           * IYZZYIXZXI
100 + 0.004802531988356293
                           * IXZZXIXZXI
101 + 0.010328819322301622
                           * IYZYIIYZYI
102 + 0.010328819322301622
                           * IXZXIIYZYI
103 + 0.010328819322301622
                           * IYZYIIXZXI
104 + 0.010328819322301622
                           * IXZXIIXZXI
105 - 0.003466391848475337
                           * YYIIIIYZYI
106 - 0.003466391848475337
                           * XXIIIIYZYI
107 - 0.003466391848475337
                           * YYIIIIXZXI
108 - 0.003466391848475337
                           * XXIIIIXZXI
109 + 0.00211113766859809
                            * IIIYYYZZYI
110 + 0.00211113766859809
                            * IIIXXYZZYI
111 + 0.00211113766859809
                            * IIIYYXZZXI
112 + 0.00211113766859809
                            * IIIXXXZZXI
113 - 0.0077644411821215335 * YZZZYYZZYI
114 - 0.0077644411821215335 * XZZZXYZZYI
115 - 0.0077644411821215335 * YZZZYXZZXI
116 - 0.0077644411821215335 * XZZZXXZZXI
117 + 0.006575744899182541
                           * YZZYIYZZYI
118 + 0.006575744899182541
                           * XZZXIYZZYI
119 + 0.006575744899182541
                           * YZZYIXZZXI
120 + 0.006575744899182541
                            * XZZXIXZZXI
121 + 0.023557442395837284
                            * IIIIIYZZYI
                           * IIIIIXZZXI
122 + 0.023557442395837284
                           * ZIIIIYZZYI
123 + 0.010889407716094479
124 + 0.010889407716094479 * ZIIIIXZZXI
125 - 0.0003518893528389501 * IZIIIYZZYI
126 - 0.0003518893528389501 * IZIIIXZZXI
127 - 0.0003518893528389501 * IIZIIYZZYI
```

```
128 - 0.0003518893528389501 * IIZIIXZZXI
129 - 0.00901204279263803 * IIIZIYZZYI
130 - 0.00901204279263803 * IIIZIXZZXI
                           * IIIIZYZZYI
131 + 0.0127339139792953
                           * IIIIZXZZXI
132 + 0.0127339139792953
133 - 0.003818281201314288 * IIIIIYIZYI
134 - 0.003818281201314288 * IIIIIXIZXI
135 - 0.003818281201314288 * IIIIIYZIYI
136 - 0.003818281201314288 * IIIIIXZIXI
137 + 0.004217284878422759 * IYYIIIYYII
138 + 0.004217284878422759 * IXXIIIYYII
139 + 0.004217284878422759 * IYYIIIXXII
140 + 0.004217284878422759 * IXXIIIXXII
141 - 0.004879740484191498 * IIYZYYZYII
                          * IIXZXYZYII
142 - 0.004879740484191498
                          * IIYZYXZXII
143 - 0.004879740484191498
144 - 0.004879740484191498 * IIXZXXZXII
145 - 0.003466391848475337 * IIYYIYZYII
146 - 0.003466391848475337 * IIXXIYZYII
147 - 0.003466391848475337 * IIYYIXZXII
148 - 0.003466391848475337 * IIXXIXZXII
149 + 0.004868302545087521 * YZYIIYZYII
150 + 0.004868302545087521 * XZXIIYZYII
151 + 0.004868302545087521 * YZYIIXZXII
152 + 0.004868302545087521 * XZXIIXZXII
153 - 0.004879740484191498 * IYZZYYYIII
154 - 0.004879740484191498 * IXZZXYYIII
155 - 0.004879740484191498
                           * TY77YXXTTT
156 - 0.004879740484191498
                           * TX77XXXTTT
                           * IYZYIYYIII
157 - 0.003466391848475337
                           * IXZXIYYIII
158 - 0.003466391848475337
159 - 0.003466391848475337 * IYZYIXXIII
                          * IXZXIXXIII
160 - 0.003466391848475337
161 + 0.004868302545087521
                           * YYIIIYYIII
162 + 0.004868302545087521
                           * XXIIIYYIII
163 + 0.004868302545087521
                           * YYIIIXXIII
164 + 0.004868302545087521
                           * XXIIIXXIII
165 - 0.004917569762418068
                           * IIIYYIIII
166 - 0.004917569762418068 * IIIXXIIIII
167 + 0.0027757462269049522 * ZIIYYIIIII
168 + 0.0027757462269049522 * ZIIXXIIIII
169 + 0.0036202487558837123 * IZIYYIIIII
170 + 0.0036202487558837123 * IZIXXIIIII
171 + 0.0036202487558837123 * IIZYYIIIII
172 + 0.0036202487558837123 * IIZXXIIIII
173 - 0.005949019975734285 * YZZZYIIIII
174 - 0.005949019975734285 * XZZZXIIIII
175 - 0.0021498576488650843 * YIZZYIIIII
176 - 0.0021498576488650843 * XIZZXIIIII
177 - 0.0021498576488650843 * YZIZYIIIII
178 - 0.0021498576488650843 * XZIZXIIIII
179 + 0.004479074568182561 * YZZIYIIIII
180 + 0.004479074568182561
                           * XZZIXIIII
181 + 0.02355744239583729
                            * YZZYIIIII
182 + 0.02355744239583729
                            * XZZXIIIIII
183 - 0.003818281201314288
                          * YIZYIIIII
184 - 0.003818281201314288
                           * XIZXIIIII
185 - 0.003818281201314288
                           * YZIYIIIII
186 - 0.003818281201314288 * XZIXIIIIII
                            * IIIIIIIII
187 + 1.0709274663656798
                            * ZIIIIIIII
188 - 0.5772920990654371
                            * IZIIIIIII
189 - 0.4244817531727133
                            * ZZIIIIIIII
190 + 0.06245512523136934
                           * IIZIIIIIII
191 - 0.4244817531727134
```

```
192 + 0.06245512523136934
                          * ZIZIIIIIII
193 + 0.06558452315458405
                         * IZZIIIIIII
                            * IIIZIIIII
194 - 0.3899177647415215
195 + 0.053929860773588405 * ZIIZIIIIII
196 + 0.06022550139954594
                           * IZIZIIIII
                           * IIZZIIIII
197 + 0.06022550139954594
198 + 0.004360552555030484 * YZZYZIIIII
199 + 0.004360552555030484 * XZZXZIIIII
                           * IIIIZIIIII
200 - 0.30101532158947975
                           * ZIIIZIIIII
201 + 0.08360121967246183
                           * IZIIZIIII
202 + 0.06278876343471208
203 + 0.06278876343471208 * IIZIZIIIII
204 + 0.053621410722614865 * IIIZZIIIII
205 + 0.010540187409026488 * IIIYYZIIII
206 + 0.010540187409026488 * IIIXXZIIII
                           * YZZZYZIIII
207 - 0.0351167704024114
                            * XZZZXZIIII
208 - 0.0351167704024114
209 + 0.010889407716094479 * YZZYIZIIII
210 + 0.010889407716094479 * XZZXIZIIII
211 - 0.5772920990654372
                            * IIIIIZIIII
                           * ZIIIIZIIII
212 + 0.11409163501020725
                           * IZIIIZIIII
213 + 0.06732342777645686
                            * IIZIIZIIII
214 + 0.06732342777645686
215 + 0.060505605672770954 * IIIZIZIIII
216 + 0.11433954684977561
                            * IIIIZZIIII
217 - 0.0011822832324725804 * IIIYYIZIII
218 - 0.0011822832324725804 * IIIXXIZIII
219 + 0.0027298828353264134 * YZZZYIZIII
220 + 0.0027298828353264134 * XZZZXIZIII
221 - 0.0003518893528389501 * YZZYIIZIII
222 - 0.0003518893528389501 * XZZXIIZIII
223 - 0.42448175317271336
                           * IIIIIIIZIII
224 + 0.06732342777645686
                          * ZIIIIIZIII
225 + 0.0782363777898523
                           * IZIIIIZIII
226 + 0.06980180803300681
                           * IIZIIIZIII
227 + 0.07055432072184756
                           * IIIZIIZIII
228 + 0.06878552428444665
                           * IIIIZIZIII
229 + 0.06245512523136934
                           * IIIIIZZIII
230 - 0.0011822832324725804 * IIIYYIIZII
231 - 0.0011822832324725804 * IIIXXIIZII
232 + 0.0027298828353264134 * YZZZYIIZII
233 + 0.0027298828353264134 * XZZZXIIZII
234 - 0.0003518893528389501 * YZZYIIIZII
235 - 0.0003518893528389501 * XZZXIIIZII
236 - 0.42448175317271336
                           * IIIIIIIZII
237 + 0.06732342777645686
                           * ZIIIIIIZII
238 + 0.06980180803300681
                           * IZIIIIIZII
239 + 0.0782363777898523
                            * IIZIIIIZII
240 + 0.07055432072184756
                           * IIIZIIIZII
241 + 0.06878552428444665
                           * IIIIZIIZII
242 + 0.06245512523136934
                           * IIIIIZIZII
243 + 0.06558452315458405
                           * IIIIIIZZII
244 - 0.00154067008970742
                           * IIIYYIIIZI
245 - 0.00154067008970742
                           * IIIXXIIIZI
246 + 0.0023679368995844726 * YZZZYIIIZI
247 + 0.0023679368995844726 * XZZZXIIIZI
248 - 0.00901204279263803
                            * YZZYIIIIZI
249 - 0.00901204279263803
                            * XZZXIIIIZI
                            * IIIIIIIIZI
250 - 0.38991776474152134
251 + 0.060505605672770954 * ZIIIIIIIIZI
                            * IZIIIIIZI
252 + 0.07055432072184756
                           * IIZIIIIIZI
253 + 0.07055432072184756
                            * IIIZIIIIZI
254 + 0.08470391802239534
                          * IIIIZIIIZI
255 + 0.05665606755281972
```

```
256 + 0.053929860773588405 * IIIIIZIIZI
257 + 0.06022550139954594 * IIIIIIIZIZI
258 + 0.06022550139954594 * IIIIIIIIZZI
259 + 0.004360552555030484 * IIIIIYZZYZ
260 + 0.004360552555030484 * IIIIIXZZXZ
261 + 0.011733623912074194 * IIIYYIIIIZ
262 + 0.011733623912074194 * IIIXXIIIIZ
263 - 0.03305872858775587 * YZZZYIIIIZ
264 - 0.03305872858775587 * XZZZXIIIIZ
265 + 0.0127339139792953 * YZZYIIIIIZ
266 + 0.0127339139792953 * XZZXIIIIIZ
267 - 0.30101532158947975 * IIIIIIIIIZ
268 + 0.11433954684977561 * ZIIIIIIIIZ
269 + 0.06878552428444665 * IZIIIIIIIZ
270 + 0.06878552428444665 * IIZIIIIIIZ
271 + 0.05665606755281972 * IIIZIIIIIZ
272 + 0.12357087224898464 * IIIIZIIIIZ
273 + 0.08360121967246183 * IIIIIZIIIZ
274 + 0.06278876343471208 * IIIIIIIIIII
275 + 0.06278876343471208 * IIIIIIIIZIZ
276 + 0.053621410722614865 * IIIIIIIIZZ
```

TERMS: 4, DEPTH:14

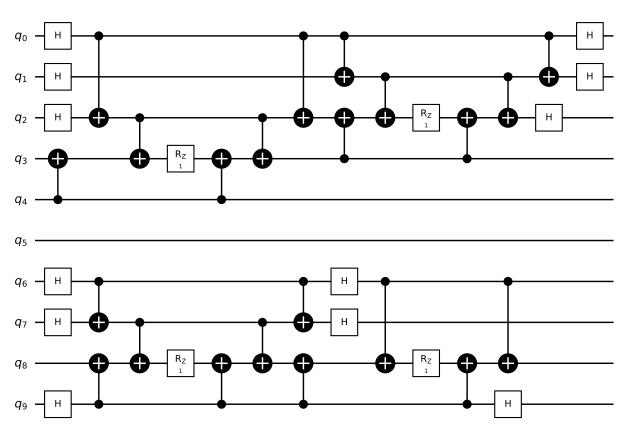


Figure 2: Example of a case in which chaining terms with high distance reduces the depth of the circuit. Here the terms are: XIXZZIIIIII, XXXZIIIIIII, IIIIIIIXXZX, and IIIIIIIZIZX.