Algorithm implementation for Quantum walk on a hypercube

Description

We implement an algorithm for a quantum walk on a hypercube from paper "Strong dispersion property for the quantum walk on the hypercube", authors of paper are Martins Kokainis, Krišjānis Prūsis, Jevgēnijs Vihrovs, Vyacheslavs Kashcheyevs, and Andris Ambainis. Our implementation is based on the description in page 2.

The walk is performed on a hypercube with 2^n vertices, where each vertex is denoted by a basis state that is formed by n qubits. We can think of a hypercube that has n dimensions, each dimension has a coordinate value 0 or 1, and only vertices that differ in one coordinate are connected (this means that each vertex is connected to n vertices).

The state space of the algorithm consists of two parts. First part is represented by n qubits, and is the space of vertices of the hypercube. The second part consists on $\lceil \log n \rceil$ qubits, where basis state represents the index of direction for the edge of a hypercube. This index points the qubit on a hypercube that is responsible for the dimension in which the edge connects two vertices (dimension, where vertices differ by coordinate – one has value 0, another has value 1).

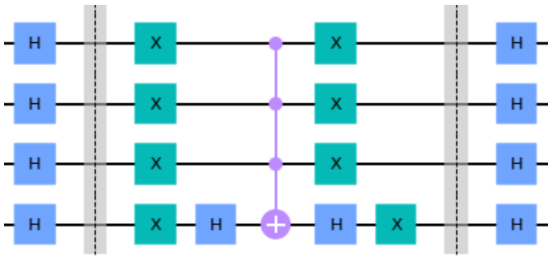
The main part of the algorithm is based on a sequence of two transitions: $W = SD_n$, where D_n is diffusion transformation, and S is shift transformation. After $t \approx 0.85n$ steps, the walker disperses over the vertices very well, with no particular vertex (basis state of the hypercube) having a substantial probability to be observed.

Implementation

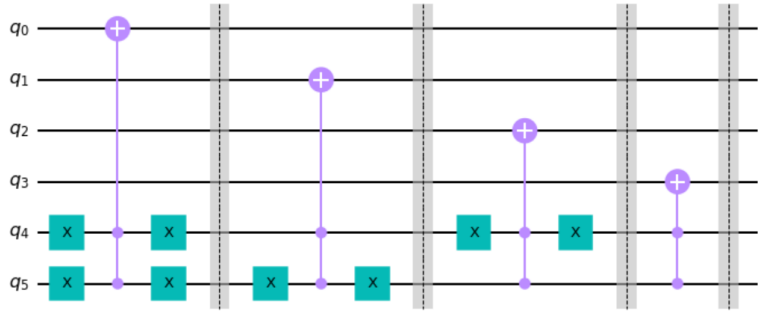
We use *qubits_for_direction* many qubits to represent directions for the edges of the hypercube. We use $2^{\text{qubits\_for\_direction}}$ for the basis states of the vertices of the hypercube.

The system is initialized in the state where qubits for direction are initialized in equal superposition, and qubits for the vertices are all initialized in the state 0. For this, we apply Hadamard gate to each qubit that is used for directions.

Diffusion transformation acts on the qubits of directions for the edges. We implement diffusion transformation in the following way: $D_n = I - 2|\psi\rangle\langle\psi|$, where $|\psi\rangle = \frac{1}{\sqrt{2^{\text{qubits\_for\_direction}}}} \sum_{i=0}^{2^{\text{qubits\_for\_direction}}} |i\rangle$. This transformation is implemented by applying Hadamard gate to each qubit before and after transformation $I - 2|0\rangle\langle 0|$. Below is example of implementation:



The shift transformation S is controlled by the qubits that represent edges (directions), and applies NOT operator (x-gate) to the qubit of the hypercube vertices, that corresponds to the index of the direction. For example, if direction represents basis state 2, then x-gate will be applied to the qubit number 2 of the hypercube. Below is example implementation for the case of two qubits for direction (and four qubits for the hypercube):



As we can see, we have four different directions, and basis state of each direction control NOT operator on a corresponding qubit of the hypercube.

To complete the implementation, we apply D_n and S repeatedly for *iteration_count* many times, and check the highest probability among all the basis states to be observed.

Experiments

We first try with *qubits_for_direction=3:*

iteration_count	Highest probability
0	1.0
1	0.125384
2	0.56245
3	0.084747
4	0.118232
5	0.022752
6	0.010645
7	0.027249
8	0.094551
9	0.094967
10	0.59137

We have $n = 8$, the theoretical minimum would be $2^{-8} = 0.00390625$, and our minimum 0.010645 has been achieved with *iteration_count=6* , which is within the results presented in the paper.

Then, we try with *qubits_for_direction=4:*

iteration_count	Highest probability
0	1.0
1	0.062965
2	0.765697
3	0.056437
4	0.347179
5	0.03342
6	0.09697
7	0.01158
8	0.016914
9	0.002462
10	0.002074
11	0.000343
12	0.000155
13	0.000097
14	0.000185
15	0.000506
16	0.001709
17	0.007256
18	0.037726
19	0.03156
20	0.249056

We have $n = 16$, the theoretical minimum would be $2^{-16} = 0.00001525878$, and our minimum 0.000097 has been achieved with *iteration_count=13* , which is within the results presented in the paper.