Description

Here we provide some explanation about content of files.

Detecting a marked vertex - 2 qubits per layer and complete binary tree.ipynb

We implement an algorithm for a quantum walk on a tree proposed by Ashley Montanaro, that allows to improve the search on a tree that is generated by backtracking algorithm. The algorithm is described in the paper "Quantum walk speedup of backtracking algorithms", Ashley Montanaro (https://arxiv.org/abs/1509.02374), page 7, Algorithm 2: Detecting a marked vertex.

The walk is based on a set of diffusion operators D_x, where for each vertex x, D_x can be implemented with only local knowledge, i.e. based only on whether x is marked and the neighbourhood structure of x. The algorithm contains two operators: R_A for the set of vertices an even distance from the root (including the root) and R_B for the set of vertices an odd distance from the root. R_A and R_B are formed by direct sums of operators D_x of according vertices.

The main part of algorithm that we implement (Algorithm 2) is Phase estimation for the operator $R_B R_A$, and if our tree contains a marked vertex, we should measure the eigenvalue 1 with high probability (with probability at least 1/2). If tree does not contain a marked vertex, the probability to measure the eigenvalue 1 is not exceeding 1/4.

We work on a binary tree with a root and n layers. Such implementation requires $2n$ qubits for encoding. Root is represented with basis state |00…00>, first layer with nodes |0…001> and |0…010>, second layer with nodes |0…00101>, |0…01001>, |0…00110>, |0…01010>, and so on. Therefore, it is possible to have a structure where basis state can encode the track of ancestors of the node, in this example with first 2 layers we see that the last 2 bits encode the parent node.

Such encoding has advantages in generation of the tree with both gate implementation and implementation of the structure of the nodes. Therefore, solution was easy to parametrize and to scale.

Detecting a marked vertex - solution without phase estimation.ipynb

Implementation is similar to the previous file, with exception that we do not implement Phase estimation. Instead, we do probabilistic sampling:

$$\frac{1}{m}\sum_{i=0}^{m-1}\left|\langle 00\ldots 0|(R_B\,R_A\,)^i|00\ldots 0\rangle\right|^2$$

This implementation allows to reduce the number of qubits, number of controlled operations and overall reduce potential number of repetitions of operation $R_B\ R_A$.

The solution is also scalable.

Phase estimation 2 layers 3 qubits without control.ipynb

Here we implement algorithm similar to the previous file. Exception is that we try to optimize the number of qubits and control operations used. As result, we have a complete binary tree with 2 layers (7 vertices total), and solution uses only 3 qubits.

DAG size estimation.ipynb

We implement an algorithm for DAG size estimation by Andris Ambainis and Martins Kokainis. The algorithm is described in the paper "Quantum algorithm for tree size estimation, with applications to backtracking and 2-player games", Andris Ambainis and Martins Kokainis (https://arxiv.org/abs/1704.06774v2), page 6, Algorithm 1: Algorithm for DAG size estimation.

The implementation of algorithm has many parts adapted from Montanaro algorithm (described in the paper "Quantum walk speedup of backtracking algorithms", Ashley Montanaro (https://arxiv.org/abs/1509.02374), page 7, Algorithm 2: Detecting a marked vertex). The algorithm is based on a set of diffusion operators D_x, where for each vertex x, D_x can be implemented with only local knowledge, i.e. based only on the neighborhood structure of x. This time basis states are denoting corresponding edges, and so D_x acts on states that describe edges, that are adjacent to corresponding vertex x. The algorithm contains two operators: R_A for the set of vertices an even distance from the root (including the root) and R_B for the set of vertices an odd distance from the root. R_A and R_B are formed by direct sums of operators D_x of according vertices.

The main part of algorithm that we implement (Algorithm 1) is Phase estimation for the operator $R_B R_A$, we obtain angle θ, and output is estimation of number of edges in the corresponding DAG: $T = \frac{1}{\alpha^2 sin^2\frac{\theta}{2}}$.

Operators D_x, R_A, R_B and Phase estimation are implemented similarly to our previous algorithm implementation.

We operate with variable δ to obtain the desired precision on our estimate of DAG size. Picked value of δ influences other parameters of the algorithm. n is distance from the root to farthest leaf (depth), T_0 is an upper bound on the number of edges (we put it as 2^{n+1}), $\alpha = \sqrt{2n\delta^{-1}}$. Then, according to the paper, we calculate $\delta_{min} = \frac{\delta^{1.5}}{4\sqrt{3nT_0}}$, and so we

determine number $bits\_of\_precision = \left\lceil \log \frac{1}{\delta_{min}} \right\rceil$ as the number of bits (qubits) of precision for Phase estimation procedure. At the end of algorithm, we receive bit values for estimate θ, convert it into decimal_value, and then calculate $\theta = 2\pi \frac{\text{decimal\_value}}{2^{bits\_of\_precision}}$. The final step is to put the value into the formula: $T = \frac{1}{\alpha^2 sin^2 \frac{\theta}{2}}$.

For complete binary tree of depth n, the number of edges should be $2^{n+1} - 2$, e.g., for $n = 2$ we have 6 edges. This is to note that the algorithm does not count additional edge that is added to the root of the tree.

We run our experiments on complete binary tree as well as trees where we remove some pairs of the leaves (starting from the right side of the tree). In our simulations, the number remove_pair_count states how many pairs have been removed. So final algorithm result is expected to be: $2^{n+1} - 2 - 2 * \text{remove\_pair\_count}$.