

Theory Days 2022

7th May

Implementation of a quantum walk on a tree and DAG size estimation algorithms for real quantum computers

University of Latvia
Faculty of computing



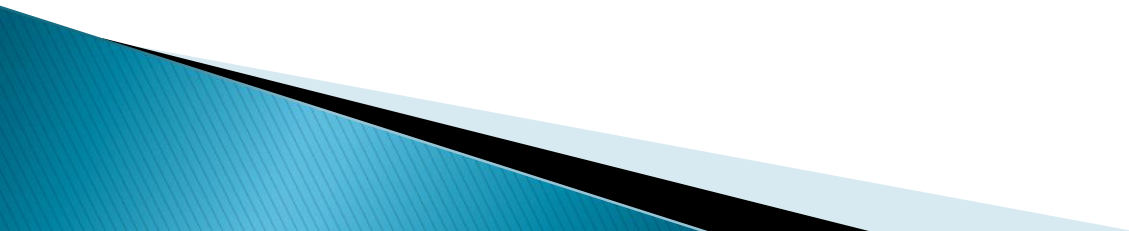
Dr. Maksim Dimitrijevič
with Prof. Andris Ambainis

What do we implement

First, we implement an algorithm called “Detecting a marked vertex” from paper “Quantum walk speedup of backtracking algorithms” by Ashley Montanaro.

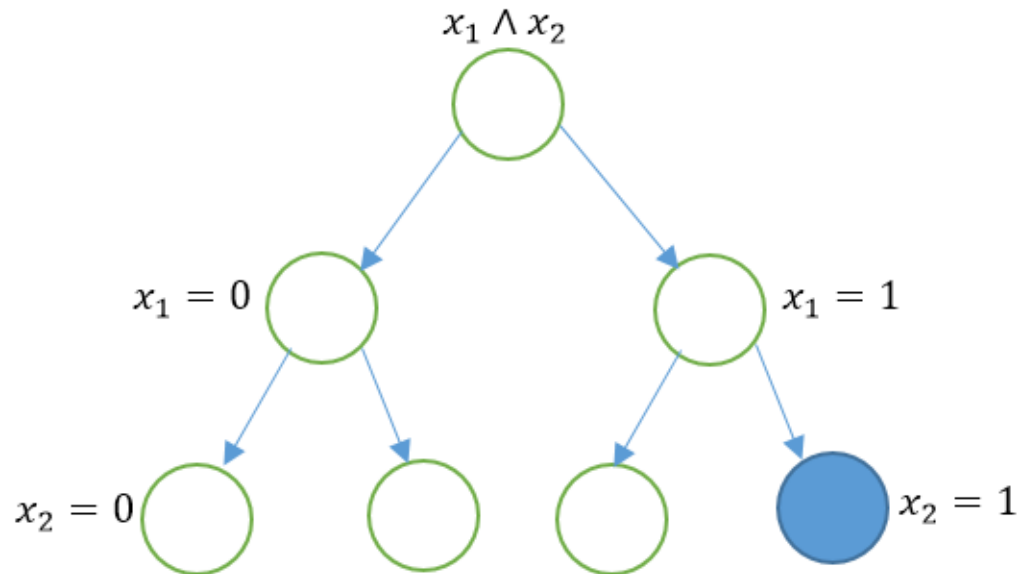
Second, we implement an algorithm for DAG size estimation from paper “Quantum algorithm for tree size estimation, with applications to backtracking and 2-player games” by Andris Ambainis and Martins Kokainis.

Detecting a marked vertex



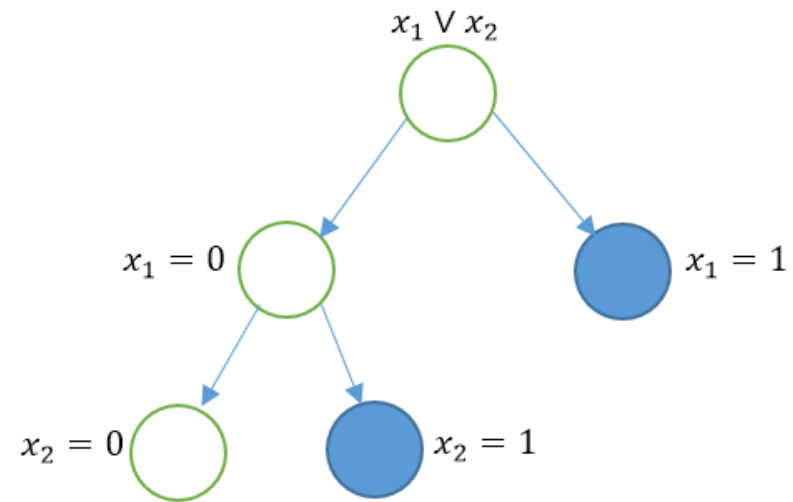
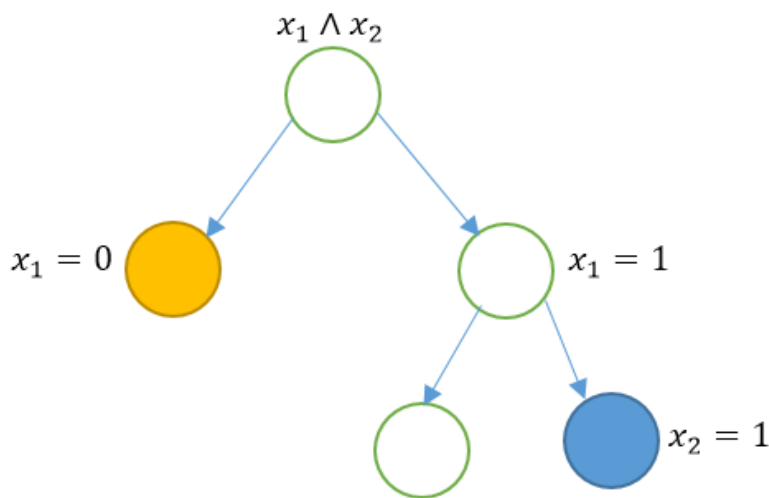
Search and backtracking

We would like to find the assignment of values of variables that satisfy our constraint satisfaction problem (CSP), e.g., SAT.



Search and backtracking

In practice, instances of CSP have some additional information about its structure. Use of backtracking allows to have more efficient classical solution of the problem than brute-force search.



What do we implement

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.

Diffusion operators

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 neighborhood structure of x .

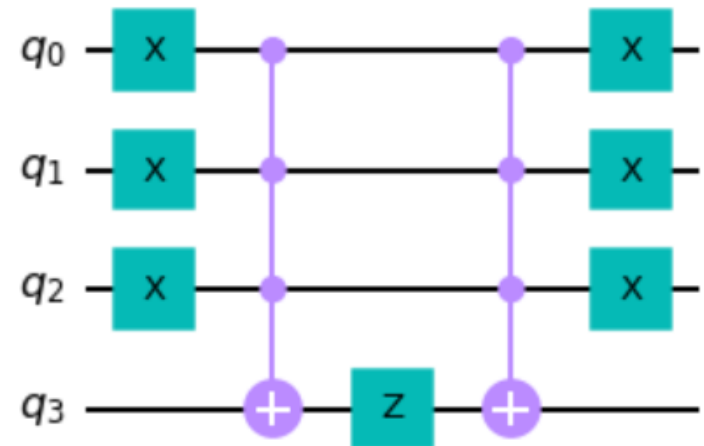
- If vertex x is marked, then $D_x = I$.
- Otherwise, $D_x = I - 2|\psi_x\rangle\langle\psi_x|$, where $|\psi_x\rangle$ depends on how many children does x has and whether x is a root.

Diffusion operators

To get $I - 2|\psi_x\rangle\langle\psi_x|$, we need to do the following sequence:

- apply a transformation that takes $|\psi_x\rangle$ to $|0\rangle$;
- apply $I - 2|0\rangle\langle 0|$;
- apply a transformation that takes $|0\rangle$ to $|\psi_x\rangle$.

How to implement $I - 2|0\rangle\langle 0|$:



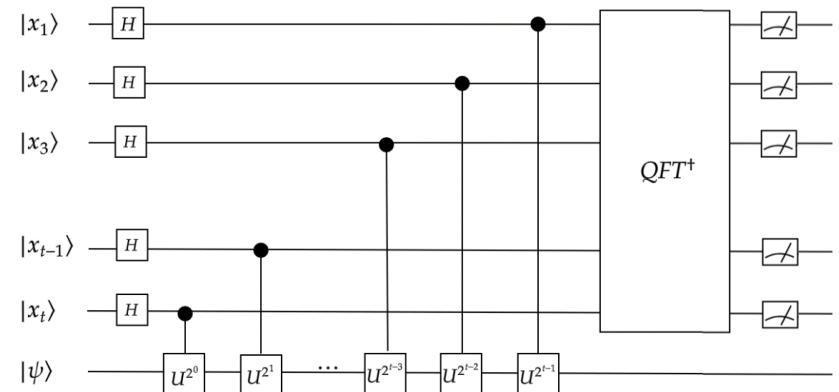
Diffusion operators

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.

Main part of the algorithm

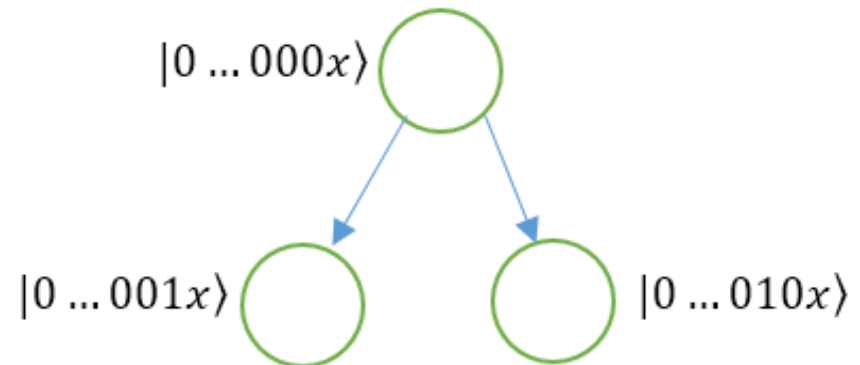
Phase estimation for the operator $R_B R_A$. If our tree contains a marked vertex, we should measure the eigenvalue 1 with high probability (with probability at least $1/2$). If tree does not contain a marked vertex, the probability to measure the eigenvalue 1 is not exceeding $1/4$.

eigenvalue is $e^{2\pi i\varphi}$
 φ is our measurement
result



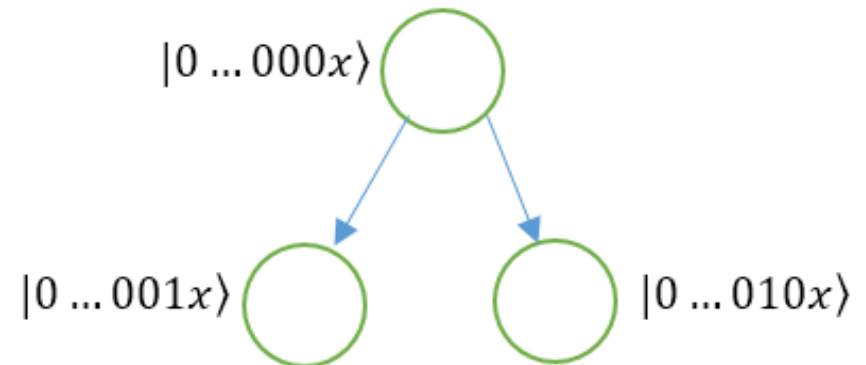
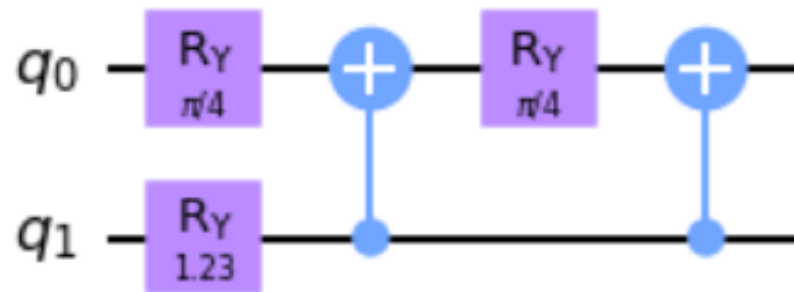
Encoding our vertices

We work on a complete binary tree with a root and n layers. Such implementation requires $2n$ qubits for encoding + 1 qubit additionally to implement $I - 2|0\rangle\langle 0|$ operation. Root is represented with basis state $|00\dots 00\rangle$, first layer with nodes $|0\dots 001\rangle$ and $|0\dots 010\rangle$, second layer with nodes $|0\dots 00101\rangle$, $|0\dots 01001\rangle$, $|0\dots 00110\rangle$, $|0\dots 01010\rangle$, and so on.

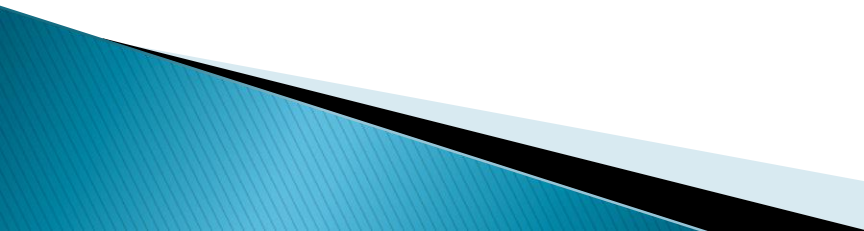


Generating $|\psi_x\rangle$

For each node x the generation of $|\psi_x\rangle$ requires just 5 gates on 2 qubits, and looks like this:



Implementation summary

- Initialize all qubits in state $|0\rangle$,
 - Apply Hadamard-gate to each qubit that will store outcomes of phase estimation,
 - Apply controlled version of $R_B R_A$ on different qubits for different number of times (according to Phase estimation procedure),
 - Apply inverse QFT to qubits that store the phase estimation,
 - Measure qubits of phase estimation part.
- 

Observations

If the root is marked, then algorithm returns eigenvalue 1 with 100% probability, does not depend on other marked vertices.

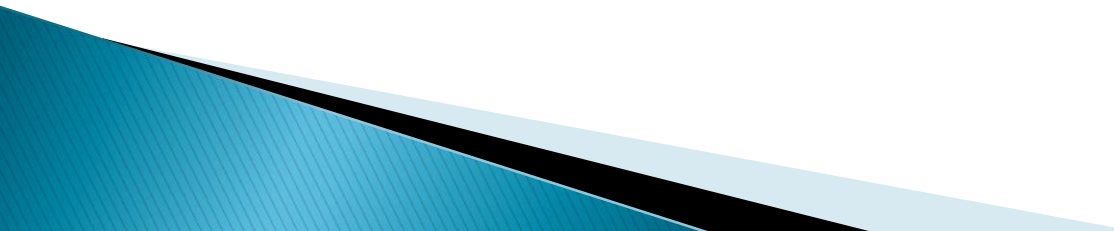
The closer the marked element is to the root, the higher is the probability to measure the eigenvalue 1.

Observations

The marked elements that are direct or indirect children of another marked element do not affect the probability (or at least affect very insignificantly). Therefore, the branch of marked elements contributes with almost the same probability as the marked element of the closest to the root element of the branch.

Observations

The more branches marked elements cover, the higher probability to observe eigenvalue 1. Our conclusion is that distance between marked vertices affects the probability (more precisely – which ancestor do the marked vertices share). The more distant the branches of marked elements are, the higher the probability.



Observations

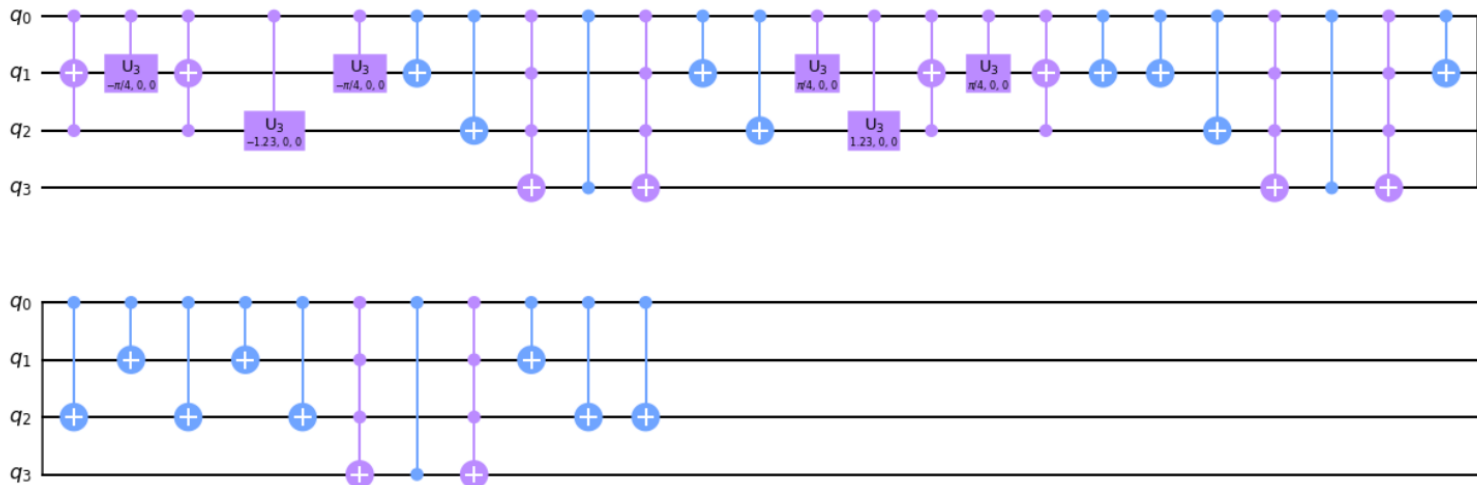
We checked the smallest probability to observe eigenvalue 1 with 1 marked element. By implementing Phase estimation with more qubits, including for smaller trees, we did not observe probability dropping below 48.8%, therefore, stated probability of $1/2$ is close to be accurate.

Observations

With trees that have at least 5 layers, the number of bits of precision in Phase estimation being equal to the number of layers in the tree gives satisfying probabilities. If we have fewer bits of precision, then in the case of no marked elements there is high probability (could be even more than 50%) to observe eigenvalue 1 after the algorithm, which makes the search procedure unreliable.

Observations

We improved the generation of the circuit for the Phase Estimation procedure – programmed controlled $R_B R_A$ manually. We achieved massive improvement in time and memory usage.



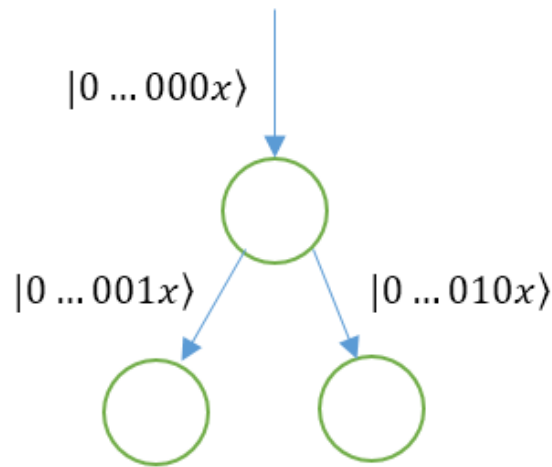
DAG size estimation

Similarity of implementation

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

Main change

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 .



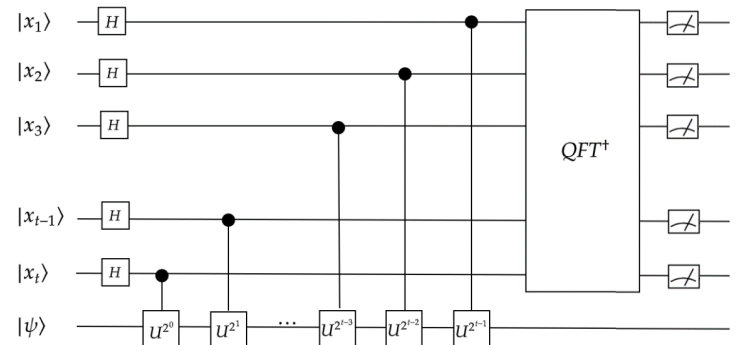
Precision

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.

If size of DAG (number of edges) is x , then we get an estimate in the interval $[x(1 - \delta); x(1 + \delta)]$ with high probability.

Parameters

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 the 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.



Parameters

At the end of algorithm, we receive bit values for estimate θ , convert it into decimal_value, and then calculate $\theta = 2\pi \frac{\text{decimal_value}}{2^{\text{bits_of_precision}}}$. The final step is to put the value into the formula: $T = \frac{1}{\alpha^2 \sin^2 \frac{\theta}{2}}$.

Experiments

- ▶ num_of_layers = 2
 - ▶ delta = 0.058 ($\left\lceil \log_{\delta_{min}} \frac{1}{\delta_{min}} \right\rceil = 10.954186184662351$)
 - ▶ bits_of_precision = 11
- remove_pair_count = 0
result: 6.022503402362584
- remove_pair_count = 1
result: 4.056179178108383
- remove_pair_count = 2
result: 2.0418953507631263

Experiments

- ▶ num_of_layers = 2
 - ▶ delta = 0.085 ($\left\lceil \log_{\delta_{min}} \frac{1}{\delta_{min}} \right\rceil = 10.127071273147157$)
 - ▶ bits_of_precision = 11
- remove_pair_count = 0
result: 5.944400519641597
- remove_pair_count = 1
result: 4.095208278612878
- remove_pair_count = 2
result: 2.0188273769201137

Picking imprecise delta

- ▶ `num_of_layers = 4`
- ▶ `delta = 0.25`
- ▶ `bits_of_precision = 9`

`remove_pair_count = 0`

`result: 33.21134408379512`

`'000000101': 4336, '111111011': 4248, '000000110': 265, '111111010': 260`

`remove_pair_count = 1`

`result: 33.2113440837946 (next closest result is: 23.06661772426938)`

`'111111011': 3002, '000000101': 2939, '111111010': 1173, '000000110': 1117`

`remove_pair_count = 2`

`result: 23.06661772426938 (next closest result is: 33.2113440837946)`

`'000000110': 2724, '111111010': 2710, '111111011': 1360, '000000101': 1301`

`remove_pair_count = 3`

`result: 23.06661772426938`

`'000000110': 4412, '111111010': 4298, '111111011': 225, '000000101': 223`

`remove_pair_count = 4`

`result: 23.06661772426938`

`'111111010': 4902, '000000110': 4715, '010011000': 62, '101101000': 62`

Picking imprecise delta

- ▶ `num_of_layers = 4`
- ▶ `delta = 0.25`
- ▶ `bits_of_precision = 9`

`remove_pair_count = 5`

`result: 23.06661772426938 (next closest result is: 16.94966819277659)`

`'111111010': 3127, '000000110': 3061, '111111001': 1061, '000000111': 1021`

`remove_pair_count = 6`

`result: 16.949668192776745 (next closest result is: 23.06661772426938)`

`'000000111': 3744, '111111001': 3741, '000000110': 568, '111111010': 560`

`remove_pair_count = 7`

`result: 16.94966819277659 (next closest result is: 12.97953319474692)`

`'111111001': 4722, '000000111': 4591, '000001000': 104, '111111000': 90`

`remove_pair_count = 8`

`result: 12.97953319474692 (next closest result is: 16.949668192776745)`

`'000001000': 3128, '111111000': 3056, '000000111': 1049, '111111001': 1029`

**Thank you for your
attention!**

