Quantum algorithms and the power of forgetting

Andrew Childs
University of Maryland

UMIACS
University of Maryland
Institute for Advanced
Computer Studies

JOINT CENTER FOR
QUANTUM INFORMATION
AND COMPUTER SCIENCE

Matthew Coudron
NIST/University of Maryland

Amin Shiraz Gilani
University of Maryland

arXiv:2211.12447 / ITCS 2023

The power of quantum computers

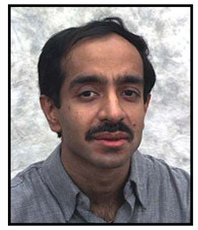
Using carefully designed interference between different computational paths, quantum computers can solve some problems dramatically faster than classical computers.

Some problems admit exponential quantum speedup.

Period finding, factoring, discrete log, quantum simulation, quantum linear algebra, Jones polynomial approximation, counting points on curves, graph connectivity with cut queries, …

Other problems admit polynomial quantum speedup.

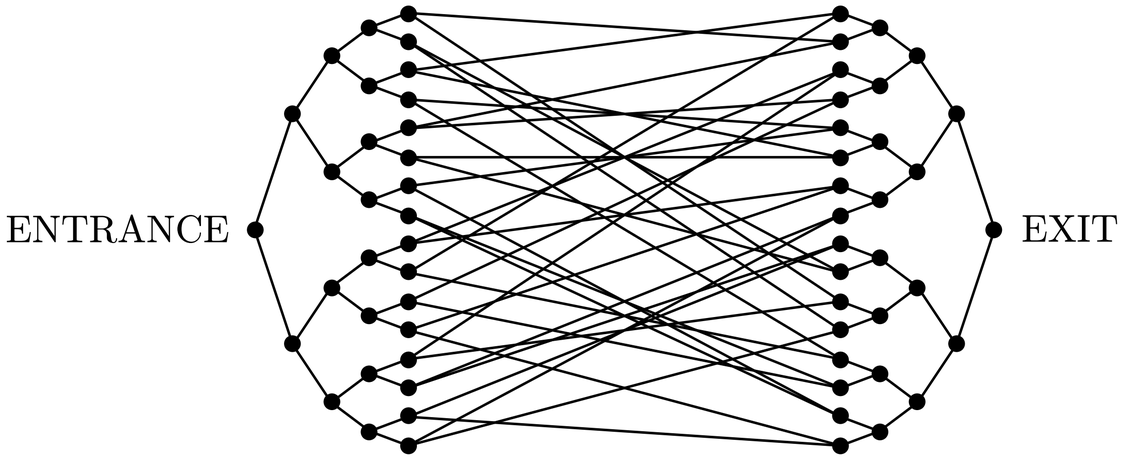
Unstructured search, formula evaluation, collision finding, network flows, finding subgraphs, minor-closed graph properties, group commutativity, convex optimization, string problems, …



What problems can be solved significantly faster by quantum computers than classical ones?

Welded tree problem

Welded tree graph:



Vertices have *names*, which are much longer than needed to name each vertex uniquely (so most strings are not the name of any vertex).

Problem: Given the name of ENTRANCE and an adjacency-list black box for the graph, find the name of EXIT.

Quantum walk from $|\text{ENTRANCE}\rangle$ stays in the *column subspace* (uniform superpositions over vertices at fixed distance from ENTRANCE).

This walk rapidly reaches a state with significant overlap on $|\text{EXIT}\rangle$.

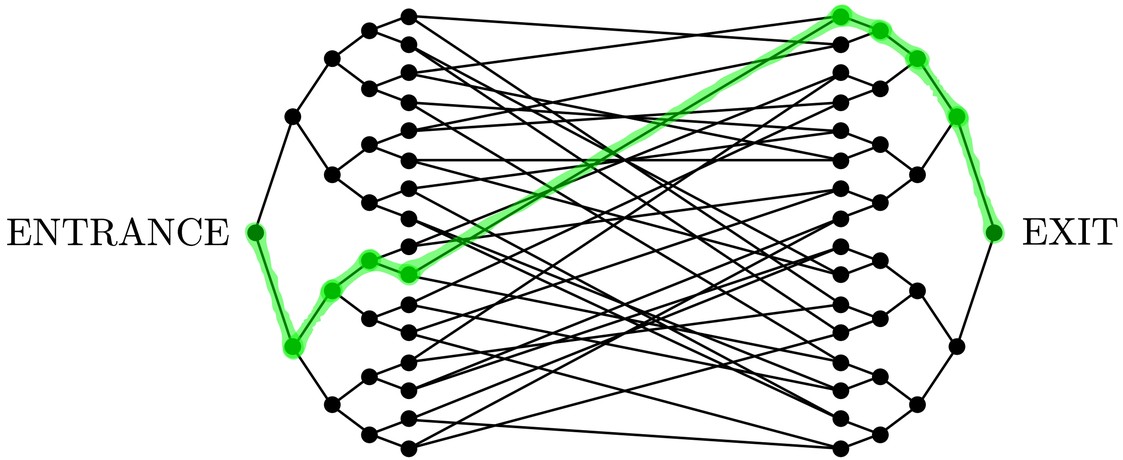
Using polynomially many queries, a classical algorithm cannot distinguish the graph from an infinite binary tree rooted at ENTRANCE, and in particular, cannot find the EXIT.

[Childs, Cleve, Deotto, Farhi, Gutmann, Spielman 03]

Path finding

Problem: Find a path from ENTRANCE to EXIT in a given welded tree graph.

A classical process can remember its history without loss of generality, so any classical algorithm for finding the EXIT implies an algorithm for finding a path.



The EXIT-finding quantum walk traverses exponentially many paths in superposition and does not output any particular path.

Modifying the quantum walk to remember a complete history of its past locations would make it effectively classical, so it would not reach the EXIT in polynomial time.

Is this tradeoff between finding and remembering a fundamental property of *all* quantum algorithms? Is there no quantum algorithm for efficiently finding a path?

Certifying a quantum computation

Many quantum algorithms uncompute intermediate results to enable interference, but this notion of forgetting a path is potentially more fundamental.

Is there a problem for which

- quantum computers provide exponential speedup, and
- there is a proof that a classical computer can use to efficiently check the solution, yet
- a quantum computer cannot efficiently produce such a proof?

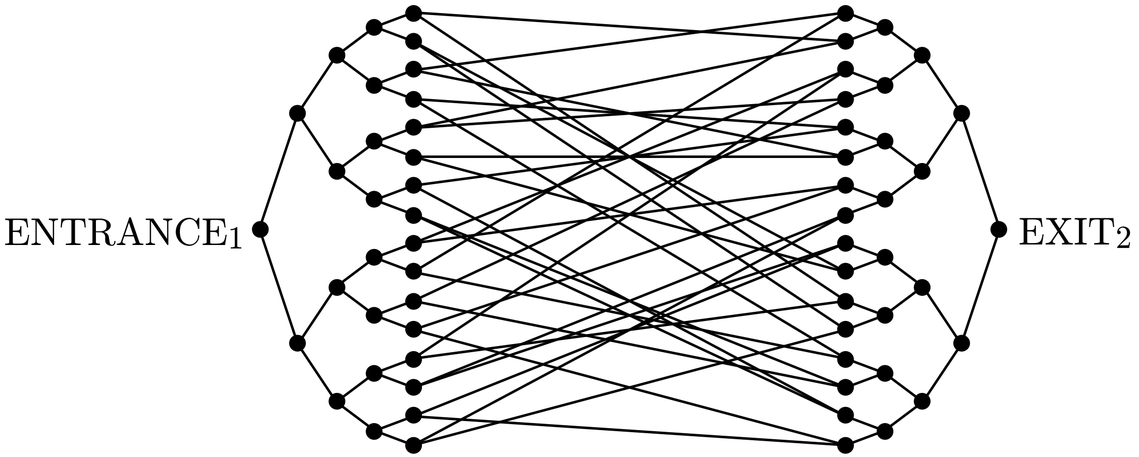
Some problems with exponential quantum speedup have certificates that a quantum computer can find efficiently (e.g., Simon's problem).

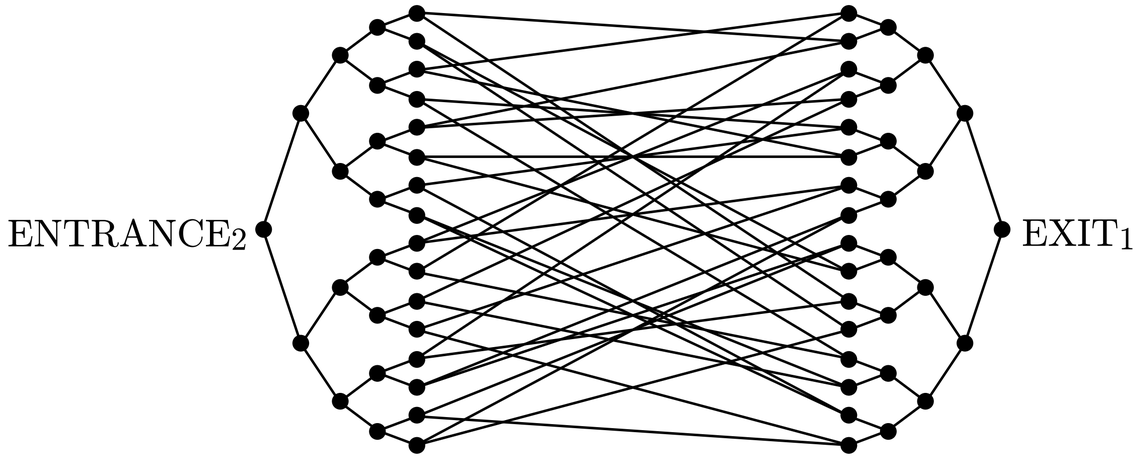
Some problems with exponential quantum speedup do not have efficiently checkable classical certificates at all (e.g., Forrelation).

Certifying the EXIT

The EXIT can easily be recognized since it and the ENTRANCE are the only degree-2 vertices.

Consider the following natural decision version of the welded tree problem: given the names of ENTRANCE_1, ENTRANCE_2, EXIT_1, and EXIT_2 for two disjoint welded tree graphs, is ENTRANCE_1 connected to EXIT_1 or EXIT_2?





An ENTRANCE-EXIT path is a classically checkable proof. Can one be found efficiently?

Snake walk

Watrous proposed a quantum walk algorithm in which the configurations are paths of a specified length in the graph ("snakes").

A suitable definition of the walk dynamics allows snakes to be explored in superposition, and still allows for the possibility of interference (though the longer snakes are, the harder it is for them to interfere).

A polynomially long snake could store a path from ENTRANCE to EXIT. Can the walk of such snakes produce a state with significant overlap on such a path in polynomial time?

Analysis by [Rosmanis 11] was inconclusive, but consistent with the possibility that the snake walk does not efficiently find a path.

Main result

We show that a quantum algorithm cannot find a path from ENTRANCE to EXIT using polynomially many queries if it is *genuine* and *rooted*.

This does not definitively rule out the possibility of an efficient path-finding algorithm, but it applies to many natural algorithms, so it significantly constrains the form such an algorithm could take.

Genuine algorithms

Informally, an algorithm is *genuine* if it only provides meaningful vertex names as inputs to the black box for the welded tree graph.

Classically, an efficient algorithm is genuine without loss of generality.

Quantumly, it is harder to define what we mean by genuine.

Main idea of the definition:

- The algorithm has *vertex registers* (storing names of vertices) and *workspace registers*
- Access to vertex registers is restricted: they can be the inputs or outputs to oracle queries; can be used to check basic properties of names (e.g., are two names equal?); or can be (partially) swapped, controlled on a workspace qubit
- Workspace qubits can be manipulated arbitrarily

The ordinary quantum walk on the graph and the snake walk are both genuine algorithms.

How could non-genuine behavior be advantageous?

Rooted algorithms

An algorithm is *rooted* if it maintains a path from the ENTRANCE to any vertex whose name it stores.

For quantum algorithms, we demand this for every computational basis state appearing in the superposition at any step of the algorithm.

Non-rooted behavior may be useful: in particular, the EXIT-finding quantum walk algorithm is not rooted.

But how could non-rooted behavior be useful for finding a path? A non-rooted path-finding algorithm would have to "forget" a path back to the ENTRANCE and later find it again.

The most natural way for a snake walk to find a path would be to evolve for a short enough time (relative to the length of the snake) so that it remains rooted.

Proof overview

Main idea: An efficient, genuine, rooted quantum algorithm can be efficiently classically simulated, up to a small error term. Since a classical algorithm cannot efficiently find the EXIT, the original quantum algorithm must not succeed.

The classical algorithm efficiently samples a "transcript" of a possible path the quantum algorithm could have taken.

This algorithm faithfully simulates the part of the quantum state that doesn't encounter the EXIT or a cycle.

We show that the remaining error term is small, essentially because an efficient classical algorithm cannot find the EXIT or a cycle (but showing this is technically involved).

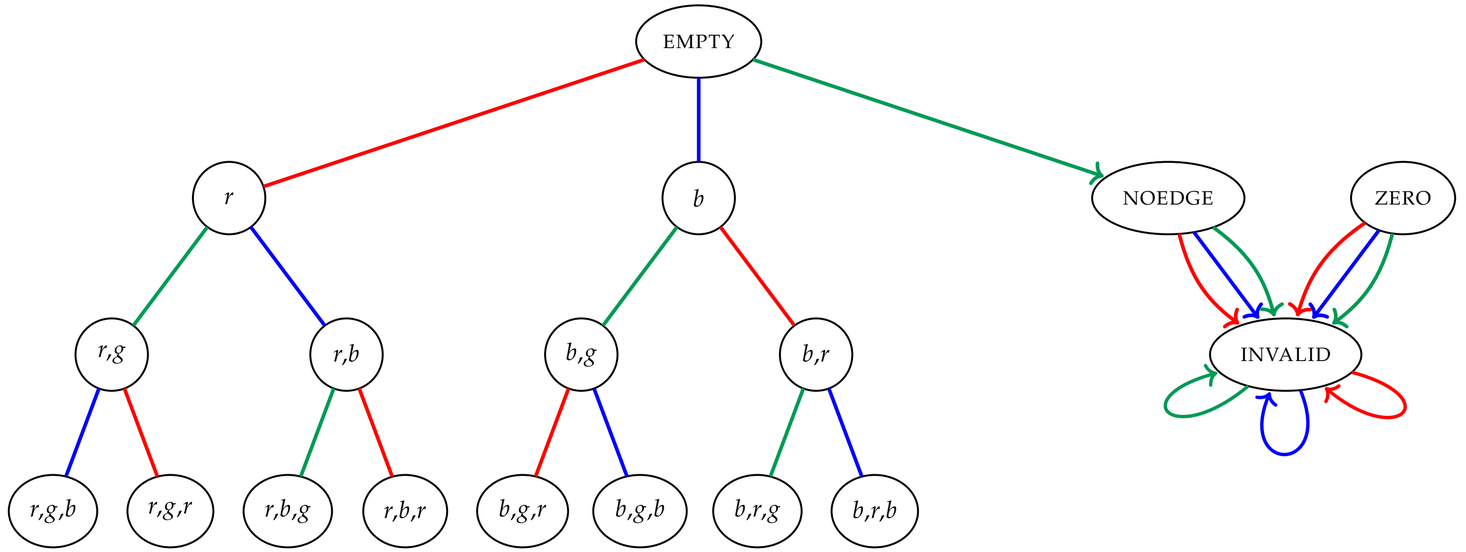
Addresses

We focus on instances in which the edges of the graph are 3-colored, and the black box respects this coloring.

An *address* is a sequence of colors that specify a vertex (together with some special addresses corresponding to special vertex names).

A vertex can have many addresses due to cycles in the welded tree graph and non-simple paths.

The *address tree* is a binary tree (plus some additional special vertices and edges) that encodes the structure of the graph that can be inferred from the addresses.



Transcript states

Given a genuine quantum algorithm, we can define a *transcript state* that captures how the algorthm would behave if run on the address tree instead of the actual welded tree graph.

A genuine algorithm operates with vertex names, not addresses, so we must specify this state by considering the algorithm's action on a given set of vertex names.

Define a mapping from each address to a distinct binary string (that could be the name of a vertex with that address). The transcript state is defined by considering how the algorithm would behave using these fictitious names for the vertices of the address tree.

Transcript states cannot capture how the algorithm would behave if it found the EXIT or a cycle. However, we show that they suffice for a good simulation.

Classical simulation

We simulate a genuine quantum algorithm operating with a 3-colored oracle as follows:

- Using two queries at the ENTRANCE, learn which color is not present there. This determines the address tree.
- Compute the transcript state of the algorithm. In general, this takes exponential time, but it does not require any queries.
- Sample a computational basis state according to the Born rule.
- Infer the addresses corresponding to the fictitious vertex names in the measured state.
- Query the actual oracle to determine the true vertex names corresponding to the measured state. If the original quantum algorithm is efficient, this requires only polynomially many queries to the oracle.

Informally, this process captures the part of the quantum algorithm that can be described by exploring an infinite binary tree. How can show that this provides a good simulation?

The good, the bad, and the ugly

We want to characterize the "good" part of the state, the part that has never encountered the EXIT or a cycle at any point in its history.

This is not straightforward to define since the state of a quantum algorithm need not have a well-defined history.

For each step of the algorithm, we inductively define
- the "good" part of the state, which has never encountered the EXIT or a cycle
- the "bad" part of the state, which just encountered the EXIT or a cycle at that step
- the "ugly" part of the state, which combines all portions of the state that have encountered the EXIT or a cycle at some point in the algorithm's history
  (the sum of all bad parts, propagated forward to the current step)

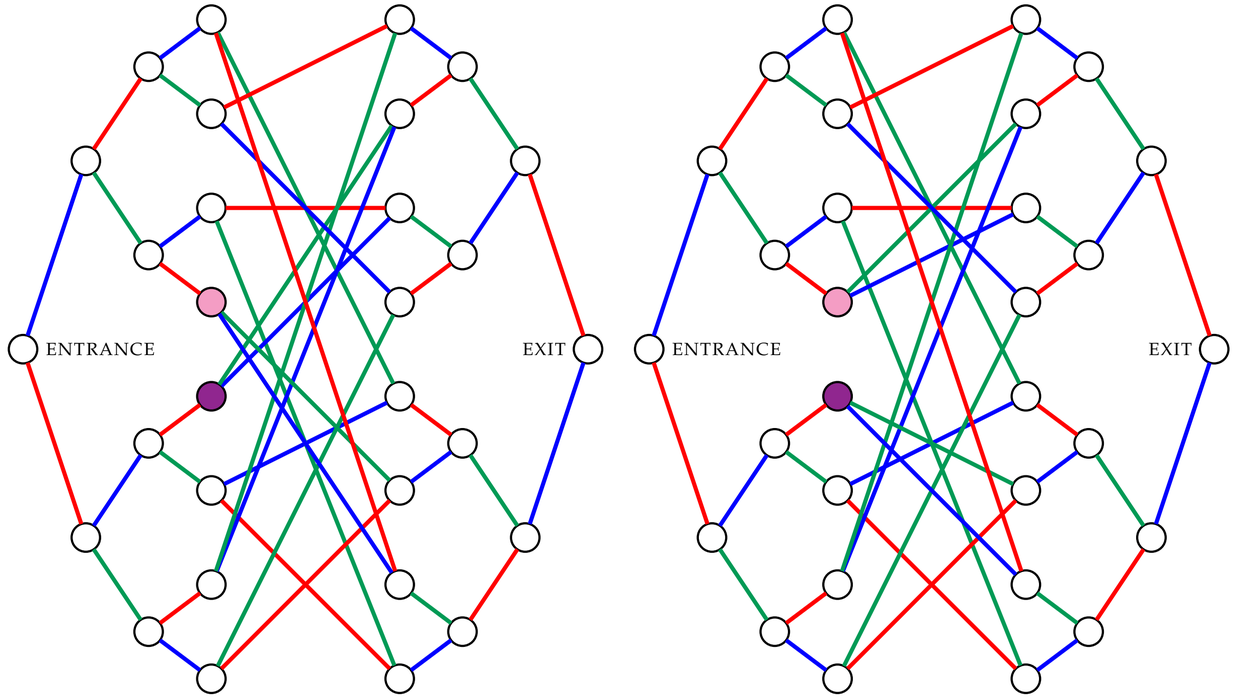
If the state at a given step has a significant bad component, then the classical simulation up to that step can be used to find the EXIT or a cycle, but we show this is not possible.

So the state is mostly good, which means the classical simulation algorithm approximates the behavior of the quantum algorithm.

Classical hardness with a 3-color oracle

The proof of [CCDFGS 03] shows that finding the EXIT or a cycle is classically hard. But we need to show this specifically for a classical algorithm that is provided with a 3-coloring of the edges of the welded tree graph.

We can randomize a 3-colored welded tree graph using a *color preserving perrmutation*, generated by swapping two left weld vertices with the same color for the edge to the left, or two right weld vertices with the same color for the edge to the right.



This randomization is sufficient to ensure that a polynomial-size subtree embedded into the welded tree graph is exponentially unlikely to encounter the EXIT or a cycle.

Conclusion

We showed that genuine, rooted quantum algorithms cannot find a path from ENTRANCE to EXIT in the welded tree graph using polynomially many queries.

Many natural approaches to the problem are genuine and rooted, so this suggests it might not be possible to efficiently find a path.

If an efficient algorithm exists, it must either use the vertex names in a non-genuine way, or forget the ENTRANCE and find it again.

Open problems

- Remove (or weaken) the assumptions of genuineness and rootedness, or (ideally) both
- Other examples of separations between the quantum complexity of detecting and finding
- Prove hardness of EXIT finding for general classical algorithms with a 3-colored oracle
- Instantiate the welded tree problem in a non-oracular setting