Accelerating quantum computing simulations

Wouter Pennings, Nico Kuijpers, Qin Zhao

What are the bottlenecks of QuantumSim and what techniques can be used to elleviate them?

Background

A quantum computer is a computer that uses quantum mechanical phenomena to solve problems too complex for "classical" computers. It is an emerging technology that has a variety of applications, such as cryptography, ML, and material science. Simulators are used to support the development of quantum technology. Each qubit doubles a quantum computer's power, making it exponentially faster but also twice as hard to simulate. Currently, QuantumSim has a practical limit of about 15 qubits.

Methodology

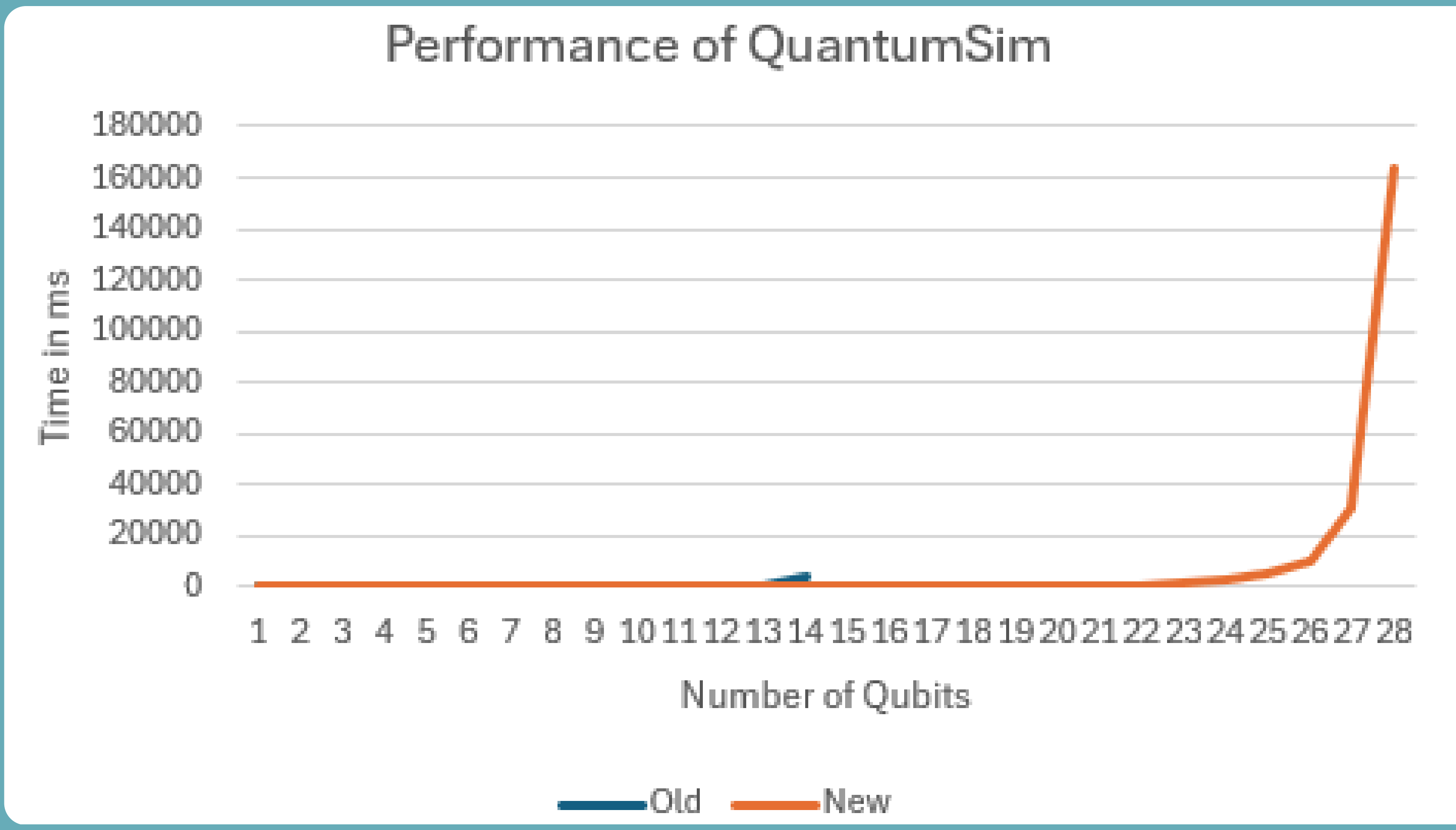
Both memory and performance bottlenecks were explored to improve QuantumSim. Using less memory allows for more simulated Qubits, and improving the performance makes the simulation run faster. Various benchmarking and profiling techniques were applied which surfaced various bottlenecks in QuantumSims implementation.

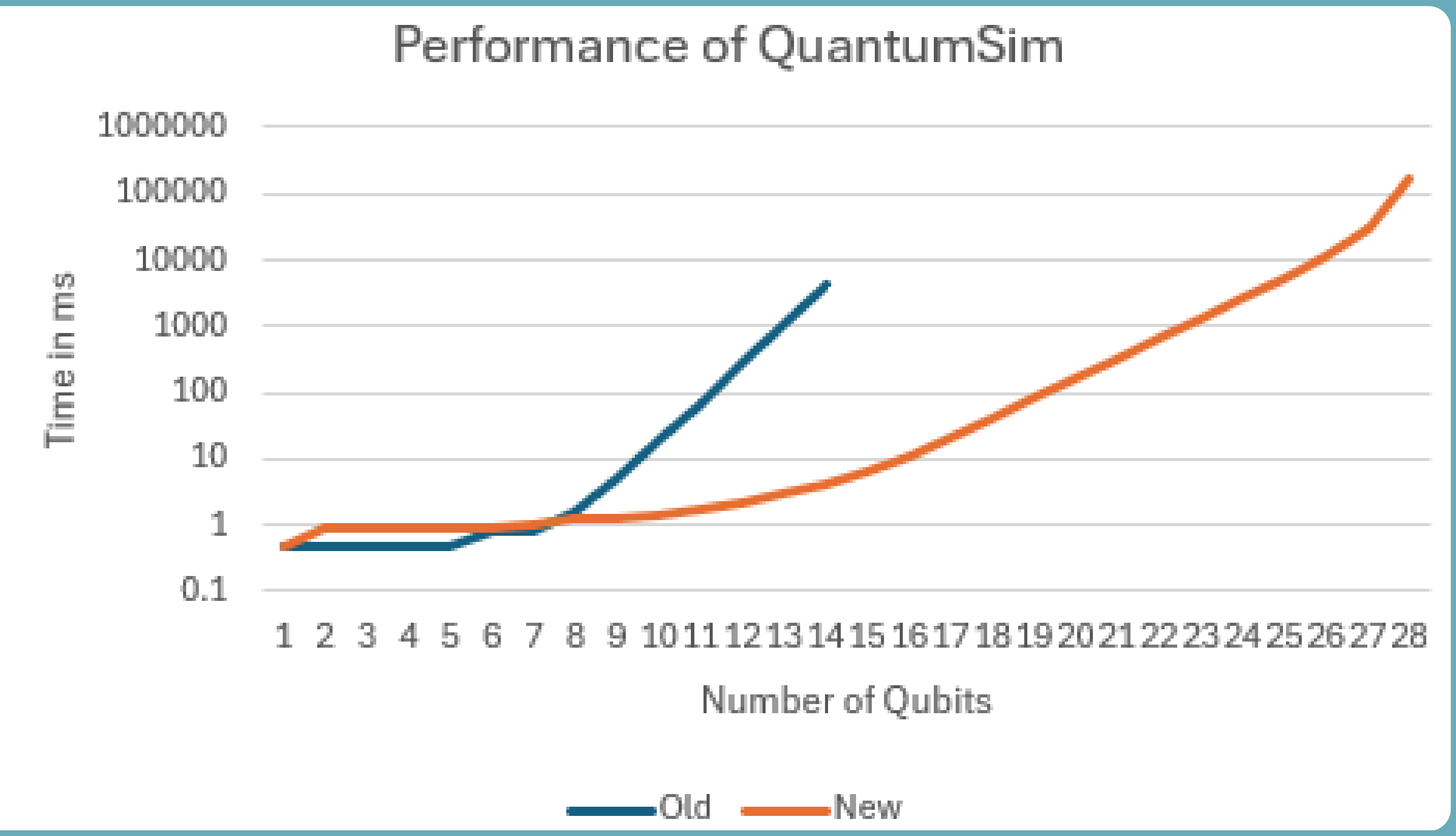
Based on previous experience and research, various improvements were A/B tested for time and space complexity to see if it was worth implementating.

Results

The main limitation of QuantumSim is the size of the unitary operation matrices. They have a space complexity of O(4^n), of which *n* is the number of qubits. However, the operation matrices are practically all zeros (99.9995% or more), meaning, a lot of computation and memory is spend on literally nothing. The data structures and algorithms were changed from dense to sparse matrices. Only the non-zero elements were stored and calculated with, both performance and memory issues were significantly improved. The two algorithms which needed to be implemented for sparse matrices were matrix-multiplication and Kronecker product.

There have also been other smaller improvements made to QuantumSim, this includes: GPU-acceleration, JIT-compilation, memory memoization and lazy matrix generation. As as result of these improvements, QuantumSim is now roughly **16.000x more memory efficient**, and **2.000x faster**.





Discussion

The outlined improvements are mostly significant for circuits which require a large amount of qubits, E.G. QAOA. These might have been not possible to run on the previous version of QuantumSim.

There is currently no known way to significantly improve the space complexity of the simulator without decreasing the precision. Two approaches possible is to store the matrices on disk instead of memory or to use smaller sized complex numbers. However, both of these approached either are a major performance hit or reduce precision.

Next steps in research

With these improvements to QuantumSim a few new research topics could be explored:

- Impact of noise on performance on new QuantumSim
- Effects of noise on quantum (hybrid) algorithms
- Scaling of QAOA algorithm on max-cut problem
- Simulated quantum annealing using the QuantumSim approach

Another possibility: Add exporting and importing OpenQASM functionality to QuantumSim

QuantumSim

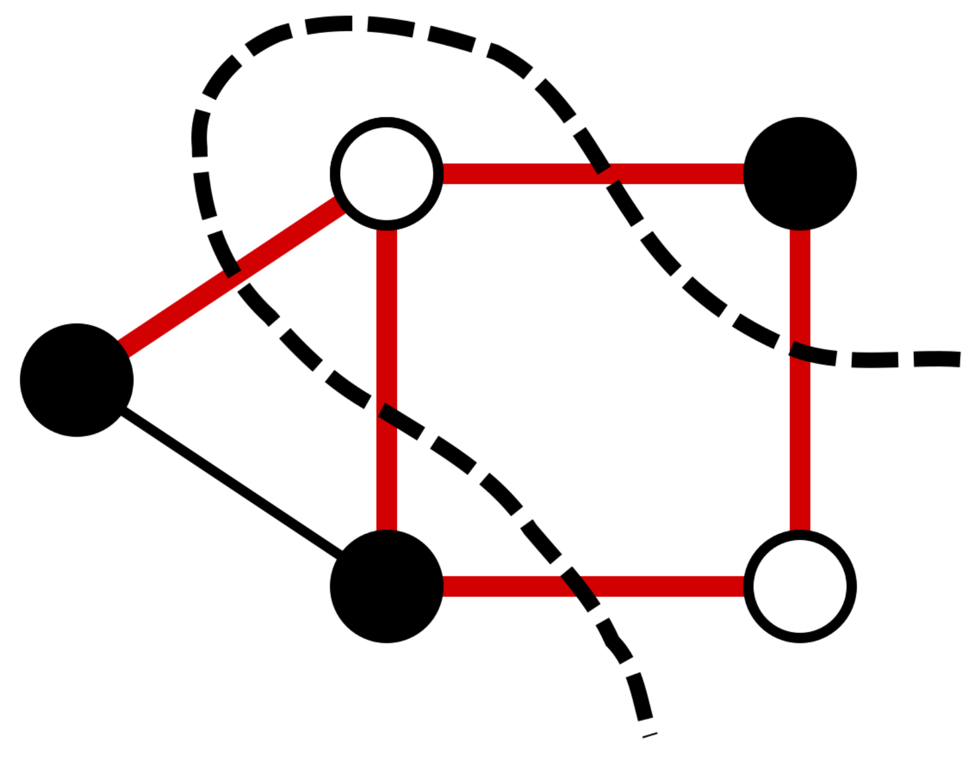
Quantum approximate optimization algorithm

Wouter Pennings, Nico Kuijpers, Qin Zhao

Background

A quantum computer is a computer that uses quantum mechanical phenomena to solve problems that are too complex for "classical" computers. QAOA is a type of algorithm that can solve optimization problems. One example of these problems is maxcut for graphs. Maxcut is a graph problem where the goal is to divide the nodes into two groups so that the number of cut edges connecting the different groups is as large as possible; the amount of cut edges is called: "cut size." In QAOA, each node in the graph is a quantum bit (qubit) in superposition, and each edge is represented by entangling two qubits. The three parameters of QAOA (β, γ, and p) are optimized using a classical method. This allows QAOA to find good approximate answers. Research is needed to understand how effective QAOA is for real-life graphs.

The result of QAOA is a list of probabilities for all possible outcomes. An outcome is a cut of a graph, and most of these cuts are not good. The bad cuts should have a low probability, but because of the nature of quantum computing (especially noisy quantum computing), that is often not the case. Having well optimized parameters is instrumental in having good probabilities coming out of QAOA



› This study aims to further understand how effective QAOA is with larger and denser graphs. It also tried to understand how noise affects the effectiveness of QAOA.

Experiments

Experiments were conducted using QuantumSim. Our QAOA implementation mirrored Fahri et al.'s approach [1]. QAOA parameters β and γ underwent SPSA optimization, while P remained fixed at five. There were two experiment types: noisy and noiseless. Across all experiments, we monitored metrics, including: average cut size with uniform probabilities, average cut size with QAOA probabilities, average cut size using the top 10, 5, and 3 QAOA probabilities, the cut size of the most probable QAOA outcome, and the brute-force maximum cut size.

Experiments type: Noiseless

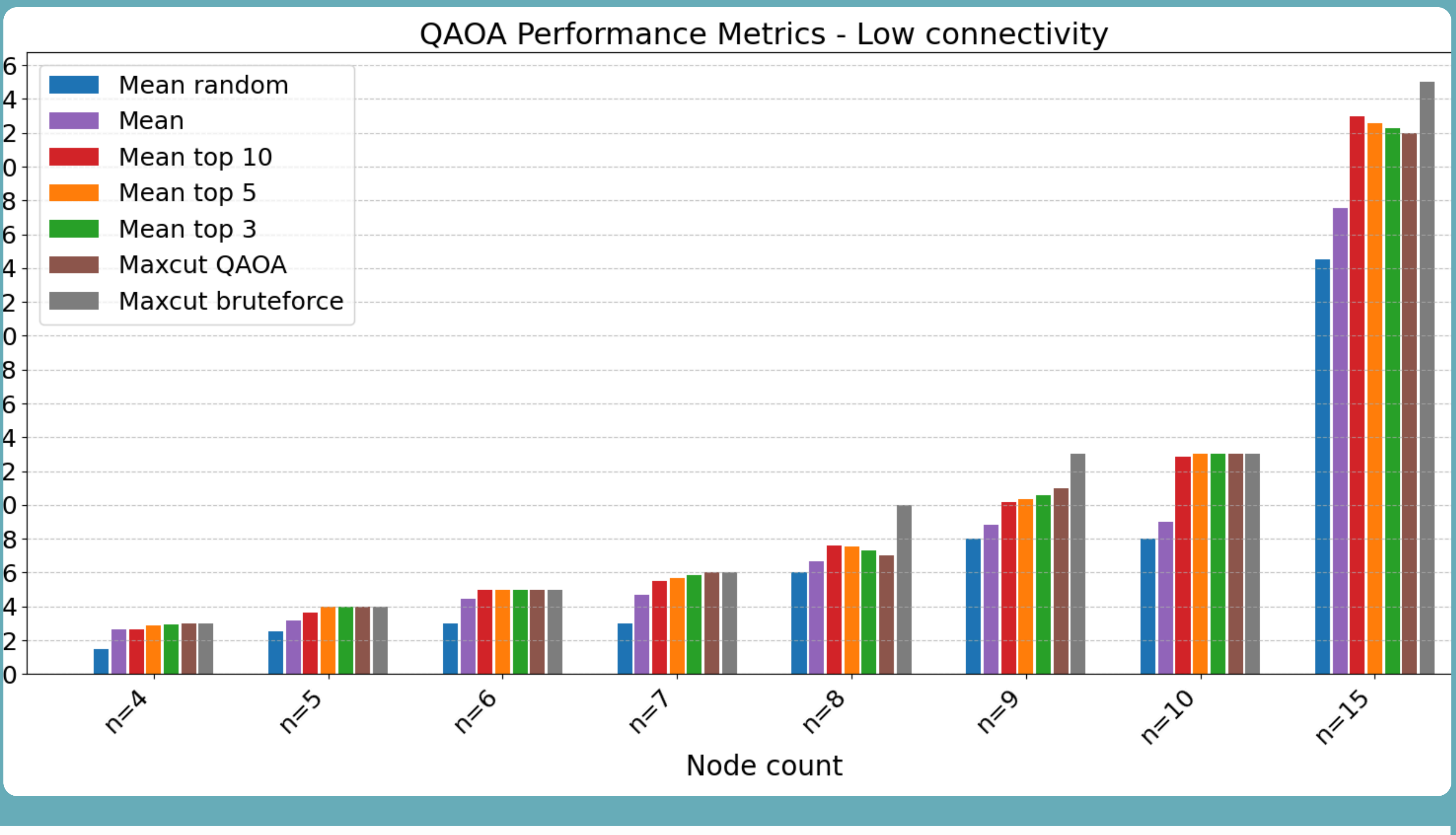
Graphs with 4, 5, 6, 7, 8, 9, 10 and 15 nodes were randomly generated in two variaties, one with little edges (sparse) and one with lots of edges (dense). QAOA, without noise, was used to approximate maxcut. The expection was to the larger and denser the graph the worse it would be at approximation a good answer.

Experiments type: Noise

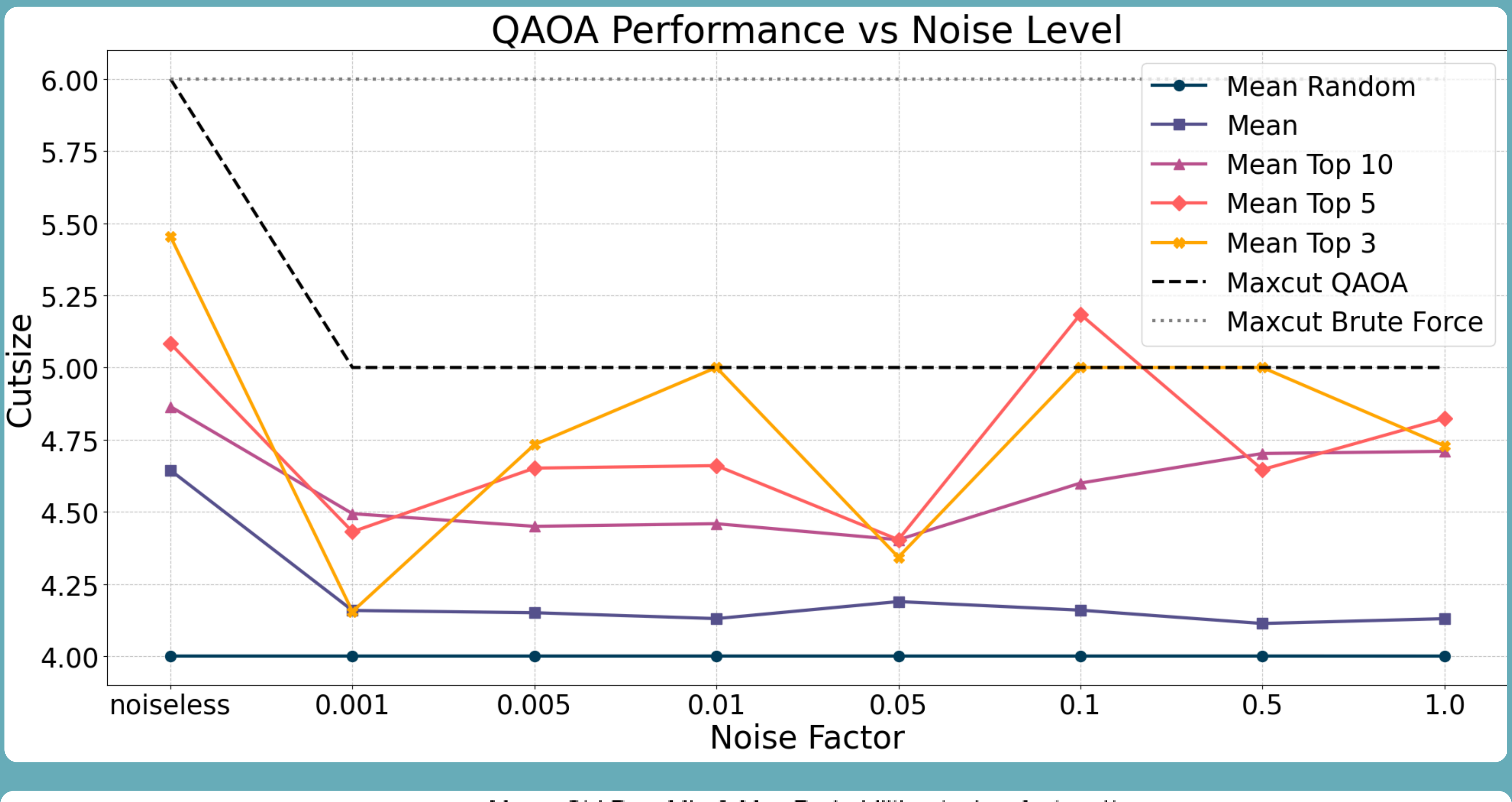
A 5-node graph in the shape of a house was used for the noise experiments. Using 7 levels of noise, from 0.1% to 100% compare to usual levels and a noiseless run as a control. The hypothesis was that the lower the level of noise, the better the approximation of maxcut.

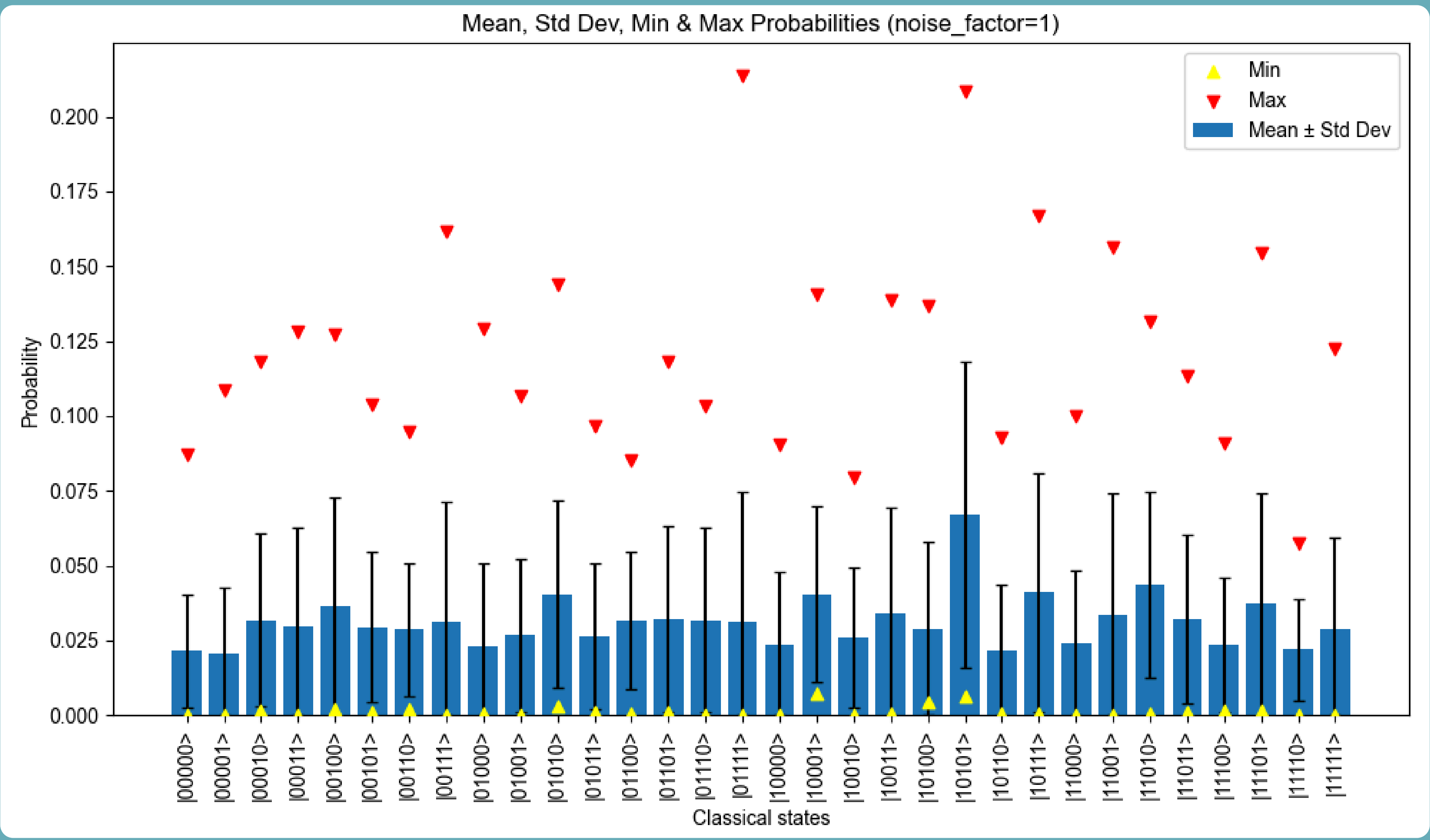
Results

Experiments type: Noiseless



Experiments type: Noise





Conclusion

In noiseless environments, QAOA consistently outperformed random selection for the maxcut problem on the graphs tested. For smaller graphs (up to 7 nodes), QAOA was effective, often finding the true maximum cut. As the graph size increased beyond 7 nodes, QAOA's ability to identify the exact maxcut diminished, regardless of graph connectivity. However, metrics still demonstrated that QAOA is relatively effective at approximating an optimal maxcut. In general, these experiments demonstrate current limitation of QAOA's scalability for larger problem instances on ideal quantum computers.

The noisy experiments provided insights into QAOA's behavior in more realistic scenarios. While the noiseless QAOA experiment found the optimal maxcut, the noisy implementations consistently resulted in a near-opptimal cut, showing a degradation from noiseless performance. A significant observation was the high variability in outcome probabilities in the presence of noise. Still, noisy QAOA performed better than random selection, demonstrating its potential even when affected by errors.

Repository to work

[1] E. Farhi, J. Goldstone, and S. Gutmann, "A Quantum Approximate Optimization Algorithm." Accessed: May 09, 2025. [Online]. Available: http://arxiv.org/abs/1411.4028