Supplementary Information for 'Sampling Frequency Thresholds for Quantum Advantage of Quantum Approximate Optimization Algorithm'

SUPPLEMENTARY METHODS

A. Quantum simulator QTensor

The most popular method for quantum circuit simulation is state-vector evolution. It stores full state vector and hence requires memory size $\propto 2^N$, exponential in the number of qubits. It can simulate only circuits with $\lesssim 30$ qubits on a common computer, and simulation of 45 qubits on a supercomputer was reported [39]. One can perform compression of the state vector and therefore simulate up to 61 qubits on a supercomputer [40].

For this work, we aim to study performance of QAOA on instances large enough to use in comparison with classical solvers. To simulate expectation values on lange graphs, we use the classical simulator `QTensor` [24; 41]. This simulator is based on tensor network contraction and allows for simulation of a much larger number of qubits. `QTensor` converts a quantum circuit to a tensor network, where each quantum gate is represented by a tensor. The indices of this tensor represent input and output subspaces of each qubit that the gate acts on. The tensor network constructed in this way can then be contracted in an efficient manner to compute the required value. The contraction does not maintain any information about the structure of the original quantum circuit, and can result in significant simulation cost reduction, not limited to the QAOA context [42].

To calculate an expectation value of some observable $\hat{R}$ in a state generated by a circuit $\hat{U}$, one can evaluate the following expression: $\langle 0|\,\hat{U}^\dagger \hat{R} \hat{U}\,|0\rangle$. This value can be calculated by using a tensor network as well. When applied to the MaxCut QAOA problem, the $\hat{R}$ operator is a sum of smaller terms, as shown in Eq. 3. The expectation value of the cost for the graph G and QAOA depth p is then

$$\begin{aligned}\langle C|C\rangle_p(\boldsymbol{\gamma},\boldsymbol{\beta}) &= \left\langle \boldsymbol{\gamma},\boldsymbol{\beta}|\hat{C}|\boldsymbol{\gamma},\boldsymbol{\beta}\right|\left.\boldsymbol{\gamma},\boldsymbol{\beta}|\hat{C}|\boldsymbol{\gamma},\boldsymbol{\beta}\right\rangle \\ &= \sum_{\langle jk\rangle\in G} \langle \boldsymbol{\gamma},\boldsymbol{\beta}|\frac{1}{2}(1-\hat{\sigma}_z^j\hat{\sigma}_z^k)|\boldsymbol{\gamma},\boldsymbol{\beta}\rangle \\ &= \sum_{\langle jk\rangle\in G} f_{\langle jk\rangle}(\boldsymbol{\gamma},\boldsymbol{\beta}),\end{aligned}$$

where $f_{\langle jk\rangle} = \langle \boldsymbol{\gamma},\boldsymbol{\beta}|(1-\hat{\sigma}_z^j\hat{\sigma}_z^k)|\boldsymbol{\gamma},\boldsymbol{\beta}\rangle/2$ is an individual edge contribution to the total cost function. Each contribution to the cost function $f_{\langle jk\rangle}$ can be evaluated by using a corresponding tensor network. Note that the observable $\hat{\sigma}_z^j\hat{\sigma}_z^k$ in the definition of $f_{\langle jk\rangle}$ acts only on two qubits and hence commutes with gates that act on other qubits. The $\langle\boldsymbol{\gamma},\boldsymbol{\beta}|$ state is not stored in memory at any time but rather is represented as a tensor network generated from the quantum circuit shown in Eq. 2. When two such tensor network representations (one for $\langle\boldsymbol{\gamma},\boldsymbol{\beta}|$ and another for $|\boldsymbol{\gamma},\boldsymbol{\beta}\rangle$) are joined aside of the observable operator, it is possible to cancel out the quantum gates that commute through the observable, thereby significantly reducing the size of the tensor network. The tensor network after the cancellation is equivalent to calculating $\hat{\sigma}_z^j\hat{\sigma}_z^k$ on a subgraph S of the original graph G.

While multiple approaches exist for determining the best way to contract a tensor network, we use a contraction approach called bucket elimination [43], which contracts one index at a time. At each step we choose some index j from the tensor expression and then sum over a product of tensors that have j in their index. The size of the intermediary tensor obtained as a result of this operation is very sensitive to the order in which indices are contracted. To find a good contraction ordering, we use a line graph of the tensor network. A tree decomposition [44] of the line graph corresponds to a contraction path that guarantees that the number of indices in the largest intermediary tensor will be equal to the width of the tree decomposition [45]. In this way one can simulate QAOA to reasonable depth on hundreds or thousands of qubits. More details of `QTensor` and tensor networks are in [24; 46; 47].

B. FLIP Algorithm

As an example of a simple and fast classical MaxCut algorithm we evaluate a local search algorithm. This class of heuristic is frequently referred to as FLIP [48] or FLIP-neighborhood [49]. A FLIP algorithm heuristic searches locally for improvements to the current solution that flip a single vertex. One still retains the freedom to choose how to search for the vertex to flip at each stage of the algorithm. Examples of vertex selection methods include randomized message passing [50] and zero temperature annealing.

The FLIP algorithm is as follows. First, for each vertex of a graph, initially assign a value of 0 or 1 at random with equal probability. From this starting point, randomly order the vertices of the graph, iterate through each vertex in order, and flip the vertex if flipping it will improve the cut value. Once all vertices have been iterated through, repeat this process until no vertices are changed in a full iteration. This procedure is analogous to zero-temperature Monte Carlo annealing and is a greedy solver. The end result is a partition in which flipping any individual vertex will not improve the cut size. On 3-regular graphs we observe that this algorithm runs on graphs of $N = 10,000$ nodes in about 70 ms on an Intel I9-10900K processor and gives a mean cut fraction of

0.847, which matches the performance of $p = 6$ QAOA.

Given more time, the FLIP algorithm can improve its performance by reinitializing with a new random choice of vertex assignments and vertex orderings, as shown in Fig. 4. Given an exponential number of repetitions, the algorithm will eventually converge on the exact result, although very slowly.

As a local binary algorithm, it runs into locality restrictions [51] and less-than-ideal performance but is extremely fast. To put this into perspective with QAOA, we implemented FLIP using Python. We observe that a simple implementation returns solutions for a 100,000 vertex 3-regular graph in < 1 second. Optimized or parallelized implementation using high-performance languages such as C++ may run several times faster. The main property is that for a graph of degree k, girth L, and size N, the FLIP algorithm runtime scales as $O(NLk)$ [50], which we verify experimentally. Notably, for any quantum initialisation step the time scaling would also be at least $O(N)$, since we have to somehow move information about the graph to the quantum device.

C. Graph statistics bounds

It is known [52; 53] that in the limit $N \to \infty$, the probability of a graph having t cycles of length l, where connectivity d is odd, is asymptotic to

$$P(l,t) = \frac{\lambda^t \exp(-\lambda)}{t!} \quad \text{where} \quad \lambda = \frac{(d-1)^l}{2l}. \tag{1}$$

Summing over all t, we find that the average number of cycles of size l is equal to the same size-independent constant

$$k_l = \sum_{t=0}^{\infty} tP(l,t) = \frac{(d-1)^l}{2l}. \tag{2}$$

This is a probabilistic estimate based on statistics of regular graphs, does not depend on QAOA, and is asymptotically precise; thus, this is a 'with high probability' (WHP) result. For 3 regular graphs, the small values of $k_l = [1.33, 2.00, 3.20$, increasing exponentially. These sparse cycles modify the subgraph that QAOA 'sees' in its local environment, as shown in Supplementary Fig. 1. Each cycle of length l modifies the subgraph of

$$m_l^p = l + l \sum_{q=0}^{p-\lceil l/2 \rceil} (d-1)^q \tag{3}$$

edges; l for the edges which participate in the cycle, plus some additional number that is exponential in p. For example, a 4-cycle of a 3 regular graph and $p = 3$ modifies 16 edges, as shown in Supplementary Fig. 1 This count of modified edges serves as an upper bound on the number of tree subgraphs

$$M_{\text{p-tree}} \geq M - \sum_{l=3}^{2p+1} m_l^p k_l, \tag{4}$$

as an edge may participate in more than one cycle. Note that $M_{\text{p-tree}}$ should trivially be ≥ 0, which occurs for large p and small N. For $p = 2$ and 3 regular graphs, this value is $M_{2\text{-tree}} \geq M - 40$, and serves as a characteristic scale for when QAOA will begin to see global structure.

The expectation value as a sum over subgraphs can then be broken into two parts: the tree subgraph and everything else. The sum can then be bounded by fixing an extremal value for the expectation value of every other subgraph, knowing that $0 \leq f_{\min} \leq f_\lambda \leq f_{\max} \leq 1$.

$$\langle \hat{C} \rangle = M_{\text{p-tree}} f_{\text{tree}} + \sum_{\lambda \neq \text{p-tree}} M_\lambda f_\lambda, \tag{5}$$

$$\leq M - M_{\text{p-tree}}(f_{\max} - f_{\text{p-tree}}), \tag{6}$$

$$\geq M_{\text{p-tree}} f_{\text{p-tree}} + (M - M_{\text{p-tree}}) f_{\min} \tag{7}$$

Combined with the lower bound of Supplemental Eq. 4, these bounds are

$$\frac{\langle \hat{C} \rangle}{M} \geq f_{\text{tree}} - \sum_{l=3}^{2p+1} \frac{m_l^p k_l}{M}(f_{\min} - f_{\text{tree}}), \tag{8}$$

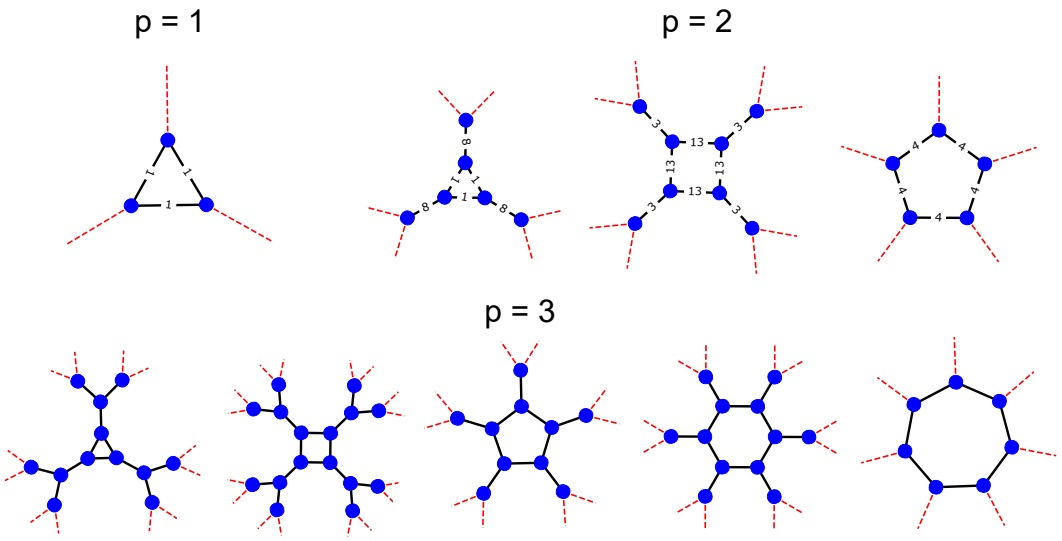
$$\frac{\langle \hat{C} \rangle}{M} \leq f_{\text{tree}} + \sum_{l=3}^{2p+1} \frac{m_l^p k_l}{M}(f_{\max} - f_{\text{tree}}). \tag{9}$$

Using the enumeration of subgraphs from [22], $f_{\min,\text{p=2}} = 0.4257$ and $f_{\max,\text{p=2}} = 0.8771$. Thus, for $p = 2$ and 3 regular QAOA, the performance can be bounded to be between

$$0.7559 - \frac{13.208}{M} \leq \frac{\langle \hat{C} \rangle}{M} \leq 0.7559 + \frac{4.848}{M}. \tag{10}$$

Therefore, for N large, the value of $\langle \hat{C} \rangle$ is bounded from above and below by a constant amount and converges to the tree value. Similarly, for N small, the value of $\langle \hat{C} \rangle$ is bounded between 0 and 1 as WHP every edge participates in at least one cycle of length $l \leq 2p+1$ and so there are no tree subgraphs to contribute to the count. In principle, these bounds may be tightened by including expectation values f_λ for more subgraphs.

This is a 'with high probability' result: there may be extremely atypical graphs that have much different numbers of tree and single cycle subgraphs. For example, an atypical graph may be one of size N that is two graphs of size $N/2$ connected by a single edge. This bound is a generalization of the work of [12], which observes that the QAOA needs to 'see the whole graph' in order to have advantage. Here the upper and lower bounds are based on the same argument, except generalized to the small-N regime



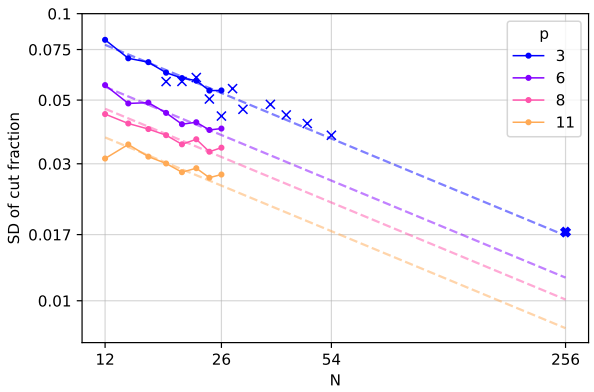
Supplementary Fig. 1. Counting types of subgraphs on sparse cycles to find upper and lower limits on QAOA expectation values. The presence of a finite number of cycles in an infinitely large graph slightly modifies the value of the QAOA expectation value by modifying the local subgraphs. As shown above, the edges that are modified as a part of each cycle for $p \leq 3$ are shown in black; vertices which connect to the rest of the graph are shown in red. Edge labels refer to subgraph indexing in [22].

D. Experimental validation of standard deviation scaling

We verify numerically the $\propto \frac{1}{\sqrt{N}}$ scaling of standard deviation using two approaches. Besides the validation of the theoretical results, these calculations allow to find the scaling coefficient to use in Equation 7.

The first approach is to calculate multiple probability amplitudes for each graph and estimate the variation by drawing a large number of samples from the distribution. This approach is feasible for graphs of small size and large p since it only requires to calculate a subset probability amplitudes. For size $N < 30$ any p is feasible since it's possible to fit the full statevector in memory. The second approach is to construct tensor network for observable $\langle C^2 \rangle$, then calculate the variance using $V = \langle C^2 \rangle - \langle C \rangle^2$. The quadratic observable has to be calculated for each pair of edges, and introduces $\propto N^2$ tensor networks. This approach allows to get the exact value of standard deviation without sampling error. Additionally, it allows to apply the lightcone optimization, which reduces computational cost for large graphs. However, the complexity grows rapidly with p and we observe that only $p = 2, 3$ are feasible on our hardware.

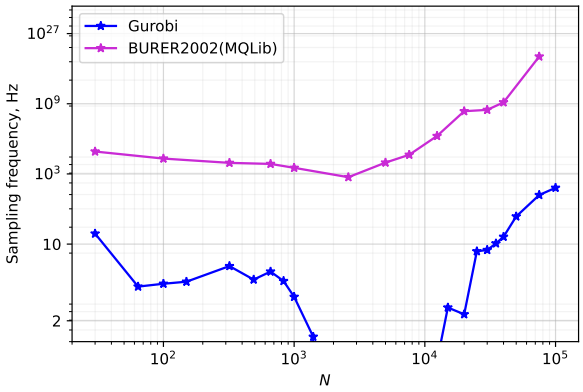
We can see that for $N \leq 26$ there is good agreement with theoretical predictions across different values of p. We use these approximate small-N results to find the scaling coefficient for each p. This will be used in the rest of the paper to estimate the standard deviation for arbitrary N and fixed p. In particular, Figure 5 uses these estimations to obtain number of multi-shot QAOA samples.



Supplementary Fig. 2. Dependence of standard deviation of MaxCut cut fraction on N and p for random 3-regular graphs. Circle markers represent approximate evaluations over 1000 samples per graph and 20 graphs for each size N. The dashed line shows a fit to the approximate data using the $\propto \frac{1}{\sqrt{N}}$ scaling. Cross markers show exact standard deviation values for larger N and $p = 3$ with one graph per each N. These values were obtained using tensor network contraction via QTensor. The bold cross marker at $N = 256$ is also an exact value of QAOA standard deviation at size corresponding to N used on Figure 4. Note that this plot has a log-log scale.

To further verify the validity of our prediction, we use exact calculations of variance via QTensor. These values are shown as crosses on Supplementary Fig. 2 and are

evaluated only for one graph for each graph size N. For small N these results are susceptible to the error due to using a single graph, since at smaller size graphs have diverse lightcones. However, as we see at larger N and $N = 256$ in particular, there is a good agreement with the predicted value of the standard deviation.



Supplementary Fig. 3. Optimal sampling frequency considering across all performance profile for classical algorithms The value is averaged over 100 seeds for Gurobi and 20 seeds for BURER2002.

E. Optimal time to compete with classical algorithms

In the main text we were comparing zero-time performance of classical algorithms and QAOA. This is justified since the classical algorithms usually improve their solution quality faster with time than multi-shot QAOA. However, this can be not the case for intermediate N where the standard deviation of the cut fraction is relatively large. This means that using a time after $t_{0,G}$ might be better for QAOA. We study this possibility by finding the sampling frequency for each point $(\Delta(t), t)$ on the performance profile for each graph. The minimum sampling frequency would match at least one point on the performance profile and thus would be sufficient for performance advantage, albeit in a very narrow regime.

We plot the average minimum sampling frequency on Supplementary Figure 3. We find that for the BURER2002 algorithm, there is no change from Figure 5, since this algorithm always scales better than the multi-shot QAOA. On the other hand, apparently it is enough to have a kHz sampling rate to match Gurobi before it starts to improve its solution. Note that the frequency for matching Gurobi can be rather noisy, since it is very sensitive to Δ. From Fig. 3 one can see that Gurobi's solution quality in region $N = 1000 \ldots 10,000$ is smaller than that of QAOA $p = 11$, a small change which could be caused by chance. This, however, results in a significantly small sampling frequency value on Supplementary Fig. 3 in the same range of N.