Automatic decomposition of quantum circuits onto small quantum machines

Benjamin Black, Onur Kulaksizoglu, Lillian Huang

December 2019

1 Introduction

In the last few years, there has been a great deal of progress in making real world quantum computers, but the number of physical qubits is remains too small to run larger problem sizes on these computers. The consequence is that the few practical uses of these computers have been problems that only require small amount of working memory to solve. The question we wish to explore relates to how to run algorithms which have larger memory requirements. In particular, there are some theoretical results that promise that if there is low global connectivity between that memory, the circuit can be decomposed into smaller, more densely connected problems that can be run with a smaller numbers of physical qubits. We wish to explore the more applied side of this area by looking at algorithms and trying to exploit features which allow lower qubit requirements.

2 Overview of Project

A primary goal of this project is to build an end-to-end system which is able to take in a representation of an arbitrary quantum circuit and accurately simulate the circuit using a quantum machine with n qubits (or a simulation of a quantum machine with n qubits).

The system will have a few components. First, we digest a description of some quantum circuit. Then, the quantum circuit will be transformed into its associated tensor network. Then, the tensor network will be partitioned into some number of connected components. The edges between these components will be transformed into classical channels using Pauli observables as described in [4], producing a large set of classical-quantum tensor networks. The tensor networks will be serialized into a sequence of quantum circuits, each of which representing one evaluation of one component. Then these quantum circuits will be run and measured in order to estimate the correct output of the original quantum circuit.

In our system, the quantum circuits will be evaluated using quantum circuit simulation software for testing purposes, but ideally, the system should be able to run on a real quantum circuit with minimal modification. We present a few initial tests as a proof of concept at the end of the report, and compare the resulting state distribution of our pipeline with the that of the non-decomposed circuit. We show that we do begin to converge to the correct state distribution within the complexity limits stated in [4].

3 Partitioning

The runtime for this decomposition algorithm is exponential in the number of edges between decomposed circuits. Therefore, it is important that we have a well-performing, relatively efficient graph partitioning algorithm for decomposing our tensor network. However, it is known that graph partitioning is NP-hard, and so we need to look for approximations and heuristic solutions for this problem. For the project we experimented with three different partitioning algorithms: a genetic algorithm, a greedy algorithm, and the METIS software package for graph partitioning [3].

3.1 METIS

Serial Graph Partitioning and Fill-reducing Matrix Ordering (METIS) is a library for graph partitioning.[3] METIS does partitions on undirected graphs and focuses on reducing the minimum cut between the different clusters while keeping a balance between the size of the clusters. It is reasonably fast and seems commonly used, that's why we picked METIS and tuned its parameters it to work with our circuits.

Objective parameter of METIS, minimum cut, suits our objective function but the load balancing was unnecessary for our case. We also needed to change it such that no sub-circuit requires more qubits than the available qubits, which is a different constraint than any classical graph partitioning parameters. Details of this modification are given in the implementation section.

3.2 Genetic algorithm

Another approach we tried was a genetic graph partitioning algorithm. Genetic algorithms are a general class of heuristic solutions to optimization problems inspired by biological evolution. Genetic algorithms are characterized by having a population of solutions which reproduce at by "crossing over" and "mutating". To keep the population size reasonable, the worst solutions are thrown away each "generation" (reproduction step). Genetic algorithms are similar to other types of general non-convex optimization techniques like simulated annealing, but are differentiated by the idea of crossover, where solutions are generated by combining two different possible solutions.

3.2.1 Algorithm details

Each solution is a list of partition labels, each of which corresponding to a node in the tensor network. The population is just a collection of K of these labelings.

- **Initialization:** Randomly assign nodes to partition labels for all K solutions.
- **Mutation:** Pick an element of the population S, and a number n of nodes to reassign. Say there are p distinct partitions in the solution S. Reassign those nodes uniformly to the p partition labels.
- **Crossover:** Randomly take two elements from the population, S_1, S_2. The output solution S is initialized with S_1. For each partition $p_n \in S_2$, assign it in to the output solution with probability $\frac{1}{2}$. There is a slight issue with the fact that similar partitioning may have very different partition labelings, so to mitigate this problem p_2 is added to S, the partition label of those nodes is assigned to be sampled from the labelings of S_1, with probability weighted by the labels in S of the nodes of p_2
- **Selection:** Partitions which use too many qubits are eliminated, and the K elements the high with smallest communication are selected.

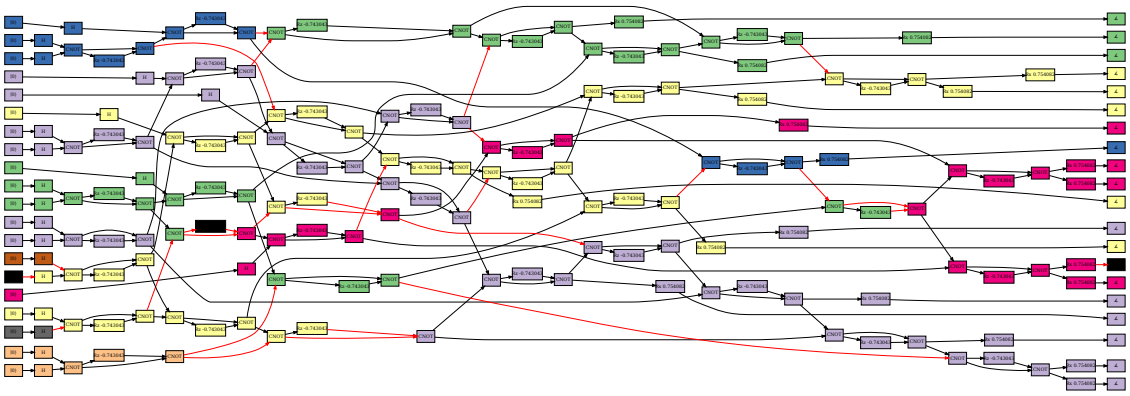
In addition, since it does not increase the communication cost to divide a partitioning into connected components, an additional step is applied after the initialization, mutation and crossover steps which divides each partitioning into connected components. This turns out to dramatically improve the solutions given, as it effectively disallows a whole class of low quality solutions.

3.3 Greedy Algorithm

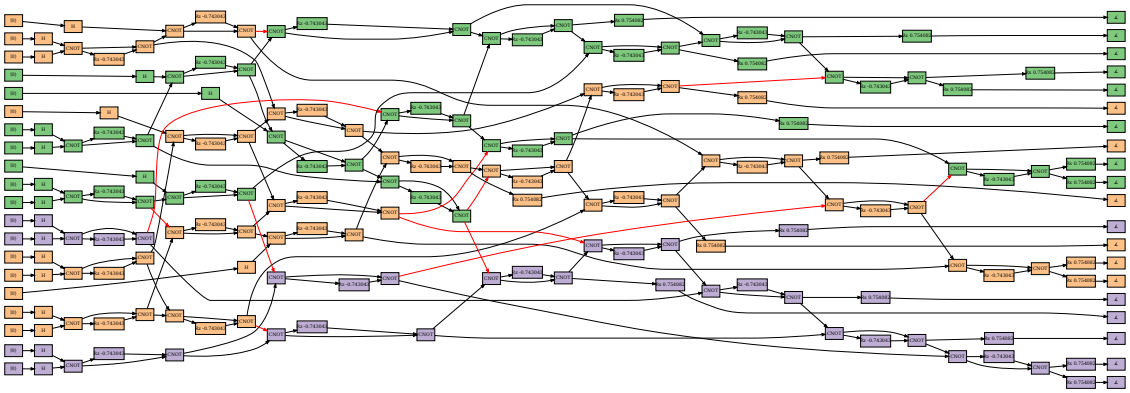
We also implemented a variant of the greedy algorithm we mentioned in the midterm report. In this approach we first reduce the network, by removing all one-qubit gates. Then we assign a separate cluster for each gate. After this initialization, in each step we merge two "mergeable" clusters with the highest score / cost.

3.3.1 Score Between the Two Clusters

The score is simply calculated by summing associativity scores between all of the neighbour nodes of the two clusters. In our case, every neighbour relation is actually a parent-child relation, since a qubit is either



(a) Output of the genetic algorithm after 100 generations.



(b) Output of the genetic algorithm after 500 generations.

Figure 1: Genetic partitioning solutions
These are outputs of the genetic algorithm run with the same parameters, but we can see that we get much better results after 500 generations than with only 100 generations (with fewer connections between separate clusters).

an input or output between two neighbour nodes.

The associativity score between two neighbour nodes considers all the children of the child node until an arbitrary depth limit, and if these children have a common qubit with the parent node, we increase the associativity score according to its distance to the parent node. Pseudo-code for associativity score is given in the Appendix; Algorithm 1.

This associativity score is designed so that it also considers the further interactions between the two nodes, so if they are going to share some qubits in the future, this increases their likelihood to merge. For example, in Fig. 2, node 3 has a higher score with node 1 than its other parent node 2. This is because the child node of node 3, the node 5, shares qubits withs node 1.

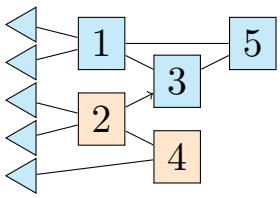


Figure 2: A case with 5-qubit simulation with 3 available qubits. After creating separate clusters for all gates, the cluster of node 3 needs to be merged and it is merged with the node 1.

3.3.2 Cost and Qubit Limit

The cost of merging the two clusters is defined as the number of required qubits for the new merged cluster minus the number of required qubits for the larger of the two smaller clusters. We also add a small cost to prevent division by 0 in the algorithm.

The number of required qubits for each cluster can be defined as $2 * m - e$, where m denotes the number of nodes in the cluster and the e refers to the number of edges between the members of the cluster.

3.3.3 Greedy Partitioning Results and Complexity

In Figure 3b, we see the partitioning of the greedy algorithm. The complexity of the algorithm is polynomial and it is quite fast. It merges almost all the nodes, and in each step it iterates through some cluster pairs to see which one has the best connections. Naively, this can be implemented in $O(n^3)$, but this can be reduced, since not all clusters are connected to each other. If we use a sparse matrix to keep cluster scores, complexity is around $O(n^2 \log(n))$, where n is the number of CNOT gates, since the only two-qubit gates we have are CNOT gates.

4 Implementation

The main deliverable in this project is the code that does this simulation. This code is on GitHub at `https://github.com/lilhuang/quantum_circuit_decomp`. At a high level, the code takes in a quantum circuit described in qTorch's qasm format [1], and a number of qubits of the target machine. It then partitions the circuit, and simulates it using the algorithm described in [4] using the QuEST library [2] to do the simulation. Exact usage is described on the README on the GitHub page.

In this section, we describe some of the details of the algorithms used.

4.1 Parameters for genetic algorithm

In the results, the following parameters were used for the genetic algorithm. Some testing suggested that the algorithm is not too sensitive to these parameters, as long as the population size, and the number of effective generations is sufficiently large. Figure 1a shows an example of what happens when the number of effective generations is too low.

$$\text{Effective Generations} = \frac{\#\text{Generations} \times \min(\#\text{Crossovers}, \#\text{Mutations})}{\text{population size}}$$

Parameter	**Value**
Population size	5000
Number of generations	5000
Mutations per generation	50
Crossovers per generation	50

4.2 Metis parameters

In an attempt to make METIS suitable for the task, a number of parameters were changed. In particular, since the communication factor is the limiting factor, not the size of the largest partition, the load balance factor was set to be higher, allowing more imbalanced loads. Also, computation time is not an issue because the graphs are very small, so parameters which trade off computation time and quality were set to strongly prefer quality.

Parameter	Value	Description
METIS_OPTION_UFACTOR	10	Allowed partition sizes are $(1 + x)/1000$, where x is the value.
METIS_OPTION_NITER	100	Specifies the number of iterations for the refinement algorithms at each stage of the uncoarsening process.
METIS_OPTION_NCUTS	100	The number of different partitionings that it will compute. The final partitioning is the one that achieves the best edgecut or communication volume.
METIS_OPTION_NSEPS	100	Specifies the number of different separators that it will compute at each level of nested dissection. The final eparator that is used is the smallest one.

Note that the balancing load factor METIS_OPTION_UFACTOR had unpredictable results, getting better results on some graphs and worse on others when set higher. And all results were still worse than the genetic algorithm. This reinforced the feeling that METIS is simply is a heuristic that is not well suited for this problem.

5 Results

5.1 Partitioning results

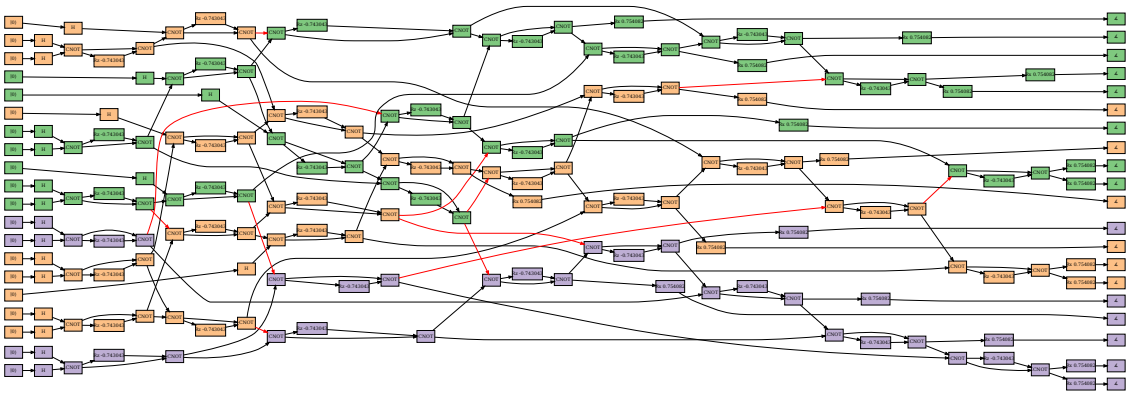
To evaluate the partitioning solutions, we took some random circuits from the qTorch "examples" folder and did some initial tests for partitioning these circuits. The results for these are illustrated in Table 1.

Table 1: Comparison of partitioning solutions on varying qubit limitations

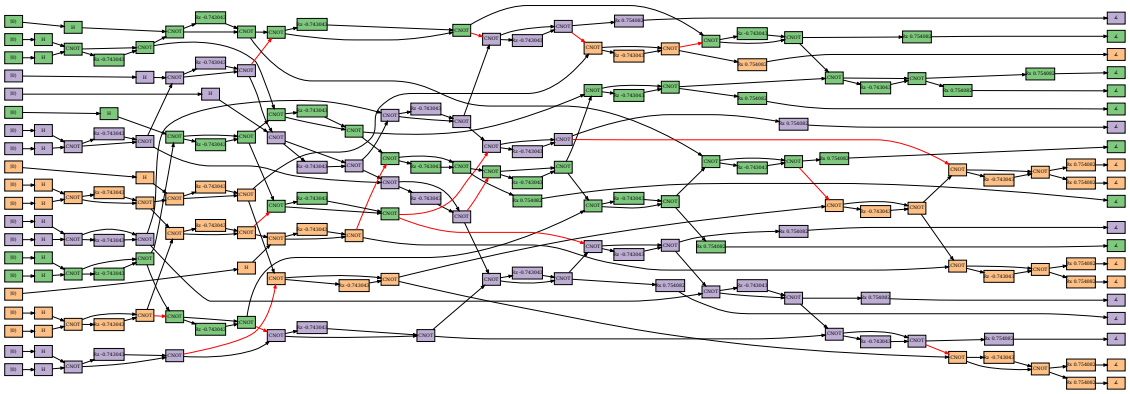
	genetic score	genetic time(s)	metis score	metis time(s)	greedy score	greedy time(s)
rand30-25	8	6.42713	11	1.9283	13	0.019721
rand20-12	12	4.78207	14	1.24236	15	0.002955
rand20-19	4	4.18873	8	1.24167	4	0.002953

Note: rand30-25 means that it is decomposing one of qTorch's random circuits with 30 qubits into subcircuits with at most 25 qubits. The score is the communication cost between the decomposed circuits.

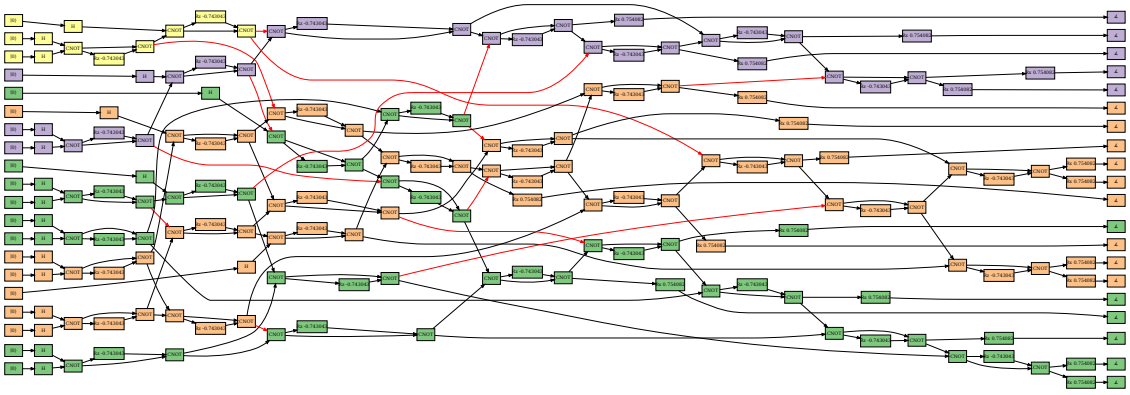
As you can see, the genetic algorithm generally performs the best of the possible solutions, but it is the slowest. The greedy strategy does very well in the rand20-19 case, where the number of qubits required is very close to the actual number of qubits, but poorly in the other cases. However, it is very fast. Metis is slowed down substantially by the parameter changes discussed in the implementation section, and also does quite poorly compared to the genetic algorithm. Visualizations of the output of these three algorithms on a particular circuit are displayed in Figure 3. Overall, it seems that to solve this NP-hard partitioning problem, using flexible strategies which give better solutions when given more computation time are required to get the best possible results.



(a) Partitioning results of the genetic algorithm.



(b) Partitioning results of the greedy algorithm.

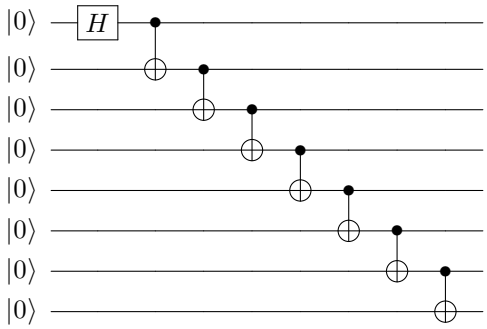


(c) Partitioning results after using METIS.

Figure 3: Visualization of various graph partitioning algorithms for the tensor network. Different colors correspond to different partitions, red lines indicate the cut edges between partitions.

5.2 Simulation results

In order to test that our implementation correctly implements the algorithm in the paper, we had to make sure the approximation bound and runtime is the same as in [4]. To test this, we took a qasm file from the qTorch "examples" folder that represents the following circuit:



The expected output state of this circuit is $\frac{1}{\sqrt{2}}|00000000\rangle + \frac{1}{\sqrt{2}}|11111111\rangle$. This circuit suits our purposes nicely, because it has a relatively simple output state distribution (if we take a measurement of the output state, half the time we should get all 0 qubits and half the time we should get all 1s), which makes it easy to verify that our pipeline converges to the correct result. Furthermore, the output state of this circuit has entangled qubits, so we know that if the final output distribution of our pipeline is correct, then it has overcome the added complication of decomposing a circuit with an entangled-qubit output.

Theorem 1 in [4] states that, once we partition a large circuit into smaller clusters with only K connections of communication between separate clusters, we should be able to achieve ϵ error after running the algorithm $O(2^{4K}/\epsilon^2)$ times, with probability $\frac{2}{3}$. Figure 4 shows how many simulations we'd theoretically need to run in order to reach a certain error bound with probability $\frac{2}{3}$ in the blue line. The orange line shows what our actual error was after running that number of simulations. We can see that our error rate stayed very close to the theoretical value, even dipping below the expected error at 10,000+ simulations.

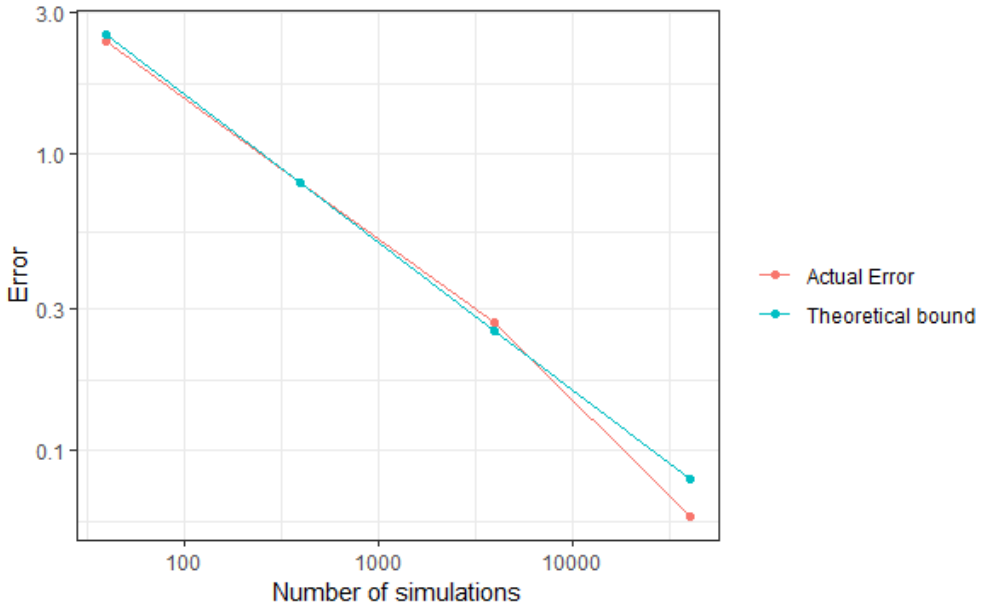


Figure 4: Expected and actual errors on different simulation sizes, according to [4].

In our example, we partitioned the above circuit into clusters of at most 4 qubits, which required $K = 2$ connections between clusters. Thus, to achieve an error of only $\epsilon = \frac{1}{3} = 0.33$, we ran the pipeline $2^8/(1/9) = 2304$ times to verify that our system does achieve this bound.

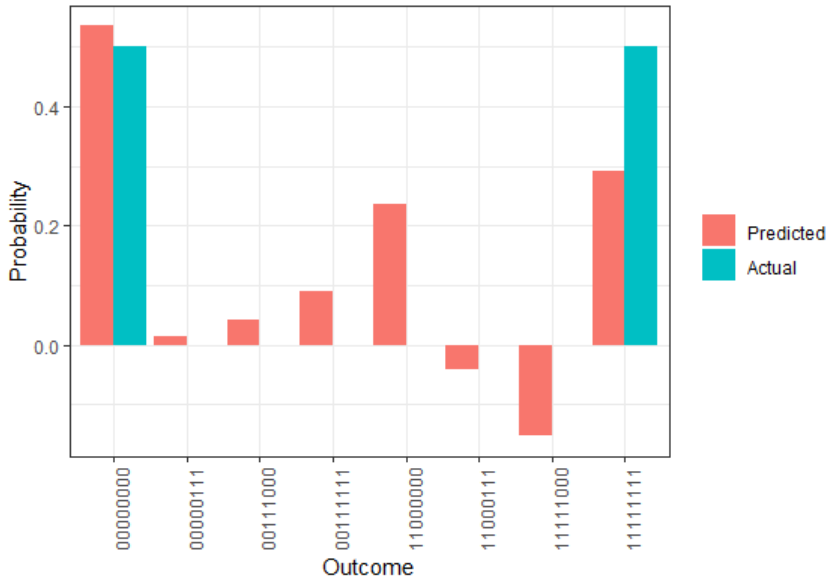


Figure 5: Probability distribution of measuring a state after running our pipeline on the given circuit, versus the expected probability distribution of measuring a state on the non-decomposed circuit.

Figure 5 shows the state distribution that we sampled after running our pipeline 2304 times. We can see that the measured state distribution (in orange) is converging to the expected state distribution (in blue). Although we measure a total of 8 states instead of only 2 as expected, we can see that the probability of measuring a state other than $|00000000\rangle$ or $|11111111\rangle$ is starting to be suppressed, while the probability of measuring those states is close to 0.5. Our final error was 0.3681, which is comparable to our goal of $\epsilon = 0.33$ but is a little high. The reason for this is because when we decompose our circuit at the K "cutting points," we do not sum over all 8^K possibilities of our circuit decomposition; instead, we randomly sample from the set 8 observables $\{O_i\}_{i=1}^{8}$ and average over them to "unbatch" the sampling process. However, although we do not exactly hit our error goal within the number of runs specified in Theorem 1 of [4], we come close enough that this small tweak in the algorithm seems to be a worthwhile change for the increased flexibility.

6 Discussion

6.1 Challenges

The problem with quantum simulation using classical information is that it takes exponential time. Though using small quantum computers to help simulate large ones does mitigate the problem to some degree (on certain problems, at least) the practicality of any exponential time method must be called into question. Once the broad questions of practicality are addressed, a real world system should make every attempt to be as efficient as possible,

6.1.1 Practicality

This project has one challenge which overshadows all the others: its practicality. There are reasons to be believe that this approach is deeply impractical. We know that there are many important problems in quantum computing which require densely connected quantum circuits to solve. Shor's algorithm is just one example. In fact, any algorithm which has an exponential time quantum supremacy over classical computers will not be efficiently simulated with classical estimations of quantum information. Thus, as the number of qubits gets a small factor larger than the physical quantum computer's, the runtime will become unreasonable. Many other problems in quantum computing, including much of quantum machine learning, are solvable in polynomial time by classical computers, so a quantum-classical system which runs in exponential time is

uninteresting.

Peng et. al. [4] suggests that one promising use case is the simulation of clustered quantum systems. This sort of system should be well suited for this approach because it decomposes at a global level because its physical clustering gives rise to only weak interactions between clusters, but it does not decompose at a local level, so purely classical computers will struggle with the simulation. In the future, our system can be tested on this case, and we can see if the results from [4] can be reproduced using our pipeline. But we also hope that there are other important problems that are suitable for this approach, as it would not be enough to only find a single use case for this algorithm.

6.2 Efficiency

When an algorithm has parts which run in exponential time, small gains in efficiency of these parts can be the difference between a practical system and in impractical one. Our project is useful in large part because, an automated system can often find structures that allow for efficiency gains even in complex problems. We tried several different approaches for partitioning and found that the best approach is to use a slow genetic algorithm that performs better when given more time and space resources, rather than a fast but fixed quality approximation.

References

[1] E. Schuyler Fried, Nicolas P. D. Sawaya, Yudong Cao, Ian D. Kivlichan, Jhonathan Romero, and Alán Aspuru-Guzik. qtorch: The quantum tensor contraction handler. *PLOS ONE*, 13(12):e0208510, Dec 2018.

[2] Tyson Jones, Anna Brown, Ian Bush, and Simon Benjamin. Quest and high performance simulation of quantum computers, 2018.

[3] George Karypis and Vipin Kumar. MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0. `http://www.cs.umn.edu/~metis`, 2009.

[4] Tianyi Peng, Aram Wettroth Harrow, Maris Ozols, and Xiaodi Wu. Simulating large quantum circuits on a small quantum computer. 2019.

A Pseudo Code for Greedy Partitioning Algorithm

Algorithm 1 Quantum Circuit Decomposer

```
procedure CALCULATE ASSOCIATIVITY(parent, child)
    arbitrary depth limit← 5
    associativity ← 0
    current level ← parent.level
    gates ← empty queue PUSH(gates, child)
    while gates not empty do
        tgate ← POP(gates)
        depth ← tgate.level - parent.level
        if depth ≤ arbitrary depth limit then
            PUSH(gates, tgate.children)
            if tgate qubit1 used by parent then
                associativity ← associativity + 2^-depth
            if tgate qubit2 used by parent then
                associativity ← associativity + 2^-depth
    return associativity
```