qTorch (Quantum TensOR Contraction Handler) Software Guide

E. Schuyler Fried, Nicolas P.D. Sawaya, Yudong Cao, Ian Kivlichan, Alán Aspuru-Guzik

September 10, 2018

The source code of qtorch can be obtained from the following URL:

https://github.com/aspuru-guzik-group/qtorch

1 License

2 Installation and Makefile

2.1 Dependencies

GCC/G++ version 4.9 or higher, Libtool or Glibtool (OSX), and GNU Make are the only external library dependencies. To check your GCC version, type `gcc -version`. If your computer runs OSX, it's likely that GCC links directly to a clang compiler. In that case, please ensure that clang is version 3.4 or higher using `clang -version`. The other dependencies, GNU make, and Libtool or Glibtool (OSX) can be installed using a local package installer such as `yum` or `brew`, or by following instructions online. However, it is likely that these packages are already installed.

2.2 Installing qTorch Using GNU Make

For convenience, an installation option is provided with the library. Please use the following commands inside the main directory qtorch for installing qTorch:

```
make install
```

or, to locally install for just one user,

```
make installlocal
```

This is necessary for running any simulations. If installation of nlopt-2.4.2 (the nonlinear optimization library used for QAOA angle optimization) fails, QAOA maxcut will not run. Next, we recommend the user set their shell environment variables as detailed in the README.txt file to be able to run the compiled executables from any directory and extend the library to their custom .cpp files. Additionally, a Makefile is provided. The available commands are: `make all`, `make qtorch`, `make test`, `make cut`, and `make clean`. The `make all` command compiles all three executables (tester, maxcutQAOA, and qtorch), and the `make clean` command removes any compiled executables. Please read the rest of the guide for further details on the other make commands.

2.3 Installing qTorch Using Docker

An alternative way of running qTorch is to run it inside a Docker container. This allows the user to use qTorch regardless of the local environment (operating system, package dependencies etc) of her machine. We provide the following Docker image on DockerHub

therealcaoyudong/qtorch

which contains the qTorch source code as well as the necessary libraries needed for building qTorch.

Here we provide a step-by-step instruction on how to use the Docker image, assuming no prior experience with Docker.

1. Install Docker on your computer. Also VirtualBox is needed;
2. In a command terminal, run `docker-machine ls`, which returns a list of virtual machines. If the list is empty, create a virtual machine called `default` by running `docker-machine create --driver virtualbox default`.
3. Configure the shell to the virtual machine by running `docker-machine env default`. Run the command suggested by the returned message.
4. Now everything is set up, run the image by `docker run -it therealcaoyudong/qtorch`. It may take a while to download the image. When the download is complete, the shell header will look like for example `root@c7d9e3be4c53#`. The hash string after @ is the identifier of the Docker container started by the `docker run` command just executed. To see a list of running Docker containers, run `docker ps`. Note that in addition to a hash string, each container is also labels with a nickname (such as happy_einstein).
5. The Docker container is effectively an Ubuntu environment with qTorch installed and executable from any directory. The source code for qTorch can be located at `/root/qtorch`. The user is free to re-build qTorch as described in Section 2.2. To see if qTorch is correctly installed, run `qtorch` anywhere inside the Docker container.

3 Introduction

The tensor contraction library provides a parallelized framework for users to simulate quantum circuits using a tensor network. Quantum circuits are translated from the QASM file format [Chu05] to a tensor network object and then contracted at the user's command, returning the probability of obtaining the measurement specified, not the amplitude. Additionally, the user must provide a list of measurements in a separate file: X, Y, Z, 0, 1, or T (Trace) on each qubit. Sample measurement files are provided for the user to peruse. Each contraction is irreversible, and the user cannot check the amplitudes of the wave-function at a specified time-step in the circuit. This is unfortunate but is also paramount in the tensor network's ability to calculate expected values for circuits with large numbers of qubits. Within the library, sample QASM files are provided for the user to look through and emulate. Furthermore, more information on how to write a QASM file compatible with the library exists in Section 2. Many gates are built into the library, but the user is free to design any arbitrary gate on one or two qubits. In this way, computation is universal, and any measurements can be executed.

4 Writing a QASM File

4.1 Basic Rules

1. Each QASM file's first line must be the number of qubits in the circuit
2. Each successive line can either act on a qubit with an operator or define a one or two qubit operator
3. A corresponding measurement file must be provided with the QASM file. If it's blank, all qubits will be traced out
4. If the QASM file is incorrect, the tensor library will not simulate it

4.2 Acting On a Qubit with an Operator

The following operators are already supported by the library: CNOT, SWAP, Hadamard, Rx(θ), Ry(θ), Rz(θ), X, Y, Z, Depolarizing Noise Channel, CRk, CZ, and CPHASE(θ). A qubit index or two qubit indices for two qubit gates come after each operator. The only operators that have extra arguments are the Rx, Ry, Rz, and CPHASE gates. Here are a few examples of operations:

- `H 0`
- `Rz 3.14159 0`
- `CNOT 0 1`
- `SWAP 1 2`
- `CPHASE 3.14159 0 1`

4.3 Defining a New 1 or 2 Qubit Operator

The software has the power to compile and execute arbitrary one and two qubit matrix operations. Note that when parsing these matrix files, the software does not check for unitary evolution. To define an arbitrary one or two qubit operator, the command def1 or def2 comes first, followed by the gate name and the path of the file the gate is located in. Here are a few examples:

- `def1 T input/t.gate`
- `def2 Cz input/cz.gate`

Within the gate file, the user must include 4 (one qubit) or 16 (two qubits) numbers separated by spaces. The numbers can either be complex and in the format (a, b), where the number is $a + bi$ or real and formatted like a regular floating point number: 3.14159. Some example gate files are included for reference. The gate should only be defined once per QASM file, and after being defined, can be used within the file like any other one or two qubit gate.

4.4 Measurement Files

A measurement file must be provided in addition to an input QASM file. A measurement file consists of `X`, `Y`, `Z`, `0`, `1`, or `T` characters separated by spaces; these characters represent projection measurements. The order of the characters corresponds to the order of the qubits, *i.e.* the first character is a projection measurement on the first qubit. If the measurement file includes 5 measurements and an 8 qubit circuit is simulated, the last three qubits will be automatically traced out. Additionally, if the file path is incorrect or the file fails to open, all qubits will be traced out.

5 The Library

The library itself contains the header files and executable files for tensor network contraction as well as supplementary libraries and executables for calculating treewidth, generating random graphs and non-linear optimization for our example of how we simulate Farhi, Goldstone, and Gutmann's quantum approximation optimization algorithm [FGG14]. The following sections will guide the user on how to utilize the library's potential.

5.1 Simple QASM simulation

The main executable file (qtorch) provided allows the user to easily simulate an arbitrary QASM file without have to create their own tensor network object. The user writes an input script file that includes the path to their QASM input file, the path to their measurement file, the contraction type they would like to use (stochastic, line-graph ordering, or user-defined). If the contraction type is line-graph ordering, the user must also provide the amount of time they would like to run the tree decomposition algorithm (quick bb) for to determine the line graph ordering. If the type is user-defined, the user must provide the path to the contraction sequence. After the user writes the script file, they can run the main executable with the provided makefile. To run, simply type:

```
make qtorch
```

and then, if the shell PATH variable has been set,

```
qtorch <global path to script file>
```

or

```
./bin/qtorch <global path to script file>
```

if the shell PATH variable was not set. Then, the output of the simulation will be printed to the console and to the file "output/qtorch.out". The user can specify a different (valid) output file as an input parameter in their script: `>string outputpath /my/path/to/output.txt`

5.1.1 Writing an Input Script

Every line in the script file begins with the character '>', otherwise, it will be ignored. To specify the QASM file path, the user types: `>string qasm` followed by the QASM file path. To specify the measurement file, the user types: `>string measurement` followed by the measurement file path. To specify the contraction method (either stochastic or line graph ordering), the user types `>string contractmethod` followed by either `linegraph-qbb`, `user-defined`, or `simple-stoch`. If the user chooses line graph ordering, the time cutoff for the ordering algorithm (Quick BB) may be specified to override the default of 20 seconds. To do that, the user types: `>int quickbbseconds` followed by the max time in seconds. If the user chooses `user-defined`, they must also specify the file which contains the contraction sequence with the line: `>string user-contract-seq` followed by the file path of the contraction sequence.

What follows are some example input scripts.

Stochastic method:

```
# Simple stochastic method
>int threads 24
>string qasm my-path/my-qasm-file.qasm
>string measurement my-path/my-measure-file.txt
>string contractmethod simple-stoch
>string outputpath my-path/output/stochastic-my-qasm.out
```

Linegraph method, full calculation all at once:

```
# Line graph decomposition method for contraction
>int threads 24
>string qasm my-path/my-qasm-file.qasm
>string measurement my-path/my-measure-file.txt
>string contractmethod linegraph-qbb
>int quickbbseconds 3000
>string outputpath my-path/output/lg-my-qasm.out
```

Linegraph method, only running quickbb and extracting the estimate for optimal ordering:

```
# Run quickbb to get approximate optimal ordering, but do not contract
>int threads 24
>string qasm my-path/my-qasm-file.qasm
>string measurement my-path/my-measure-file.txt
>string contractmethod linegraph-qbb
>int quickbbseconds 3000
>bool qbbonly true
```

Linegraph method, contraction only, reading the approximate optimal ordering from a previous run:

```
# Contract tensor network from previously found approximate optimal ordering
>int threads 24
>string qasm my-path/my-qasm-file.qasm
>string measurement my-path/my-measure-file.txt
>string contractmethod linegraph-qbb
>bool readqbbresonly true
>string outputpath my-path/output/prevrun-my-qasm.out
```

5.1.2 User-Defined Contraction Sequence Files

The user defined contraction sequence is specified by a wire elimination ordering, and each wire is defined by a pair of nodes. Therefore, each line the the user defined contraction sequence file contains two numbers separated by spaces, which represent the indices of the nodes connected by the wire. The indices of the nodes are specified as follows: the first n nodes for an n qubit simulation specify the initial state nodes, and the last n nodes specify the measurements. The other nodes, which are gates in the circuit are numbered by the ordering specified in the QASM file. A quick example ordering for a 2 qubit network with one 2 qubit gate would be: `0 2/n 1 2/n 3 2/n 4 2/n`

5.1.3 Line-Graph Contraction and QuickBB Ordering

The Line-graph contraction method runs a binary executable in the library: `quickbb_64` by default to determine the optimal wire elimination ordering of the tensor network. This can be changed to run `quickbb_32` for a 32 bit system by inserting the line: `>bool 64bit false` into the input script file. To be able to run either executable, the user's system must be GNU/Linux and able to execute ELF executables. If this is not the case, we recommend using the library on a different operating system or sticking to the stochastic contraction method.

5.1.4 Modify the Threading for CPU Optimum Efficiency

The default number of threads used for large tensor contraction is eight, but the user has the option to modify this parameter for use on a supercomputer or a computer that supports more than eight threads. To change the number of threads, the user simply adds: `>int threads <x>` where <x> is the new number of threads to use. All threading is done via the C++ standard library's threading class, which uses pthread.

5.1.5 Understanding the Simulation Output

The output of the simulation, whether it's printed to the console, the default output file: "output/qtorch.out", or a customized output file path, is straightforward. If the simulation fails, any errors will be printed and the simulation aborted. If the simulation succeeds, the output file will contain three lines. The first line is the output of the contraction as a complex number (a, b), where the resulting amplitude is $a + b * i$. It's important to note that this number is the probability of reading the measurement string, not the amplitude of the string, as the simulation stores all values in density matrix format. The second line is the number of floating point operations that it took to contract the entire tensor network. This is for performance documentation if necessary. The final line is the time the simulation took in seconds, enclosed in brackets.

5.2 More Advanced QASM Simulation

To perform more advanced QASM file simulation, i.e. simulate a batch of files or modify simulation parameters, the user must write their own code and integrate the objects provided with the library under the qtorch namespace. We provide examples of how this can be done as a supplement to this guide. After reading, we encourage the user to try their own implementations or mimic our examples.

5.3 Header Files

First, it's important for the user to orient themselves with all the files in the library.

5.3.1 Network.h

The Network header file contains the Network object class. Each Network object represents an entire tensor network, comprised of both Wires (connections between tensors and Nodes (tensors), but the Network only stores pointers to all of the Nodes. The class also contains functions that parse the QASM and measurement files, contract two tensors, reduce the circuit to only non-adjacent two qubit gates, print the circuit to graph files formatted for visualization or treewidth

calculations, get the final value from the simulation, and reset the circuit. For exact semantics, we suggest you look at the source code.

If the user would like to cut off a simulation that takes too long or restrict the simulation time, they can access the global variables: totTimer and maxTime located within the network class. Both variables are non-static and must be set for each different instance of a Network class. To use the timer, the user sets the value of maxTime to the maximum simulation time (in seconds). Then, the user starts the totTimer and calls one of the contraction methods. The contraction will automatically return after the maximum simulation time.

5.3.2 Node.h

The Node.h file contains the Node class and all of its inheritors. Each instance of the Node class is a tensor in the tensor network. The Node stores the data held in the tensor in a vector of complex doubles, and it stores connections to other nodes in the form of Wires. Therefore, when a Network parses a circuit, each gate becomes a Node. Therefore, an inheritance hierarchy exists in the Node class that helps create a CNOTNode when there's a CNOT in the QASM file or an HNode if there's an H in the QASM file. Once two tensors are contracted, their inner product becomes an IntermediateStateNode. The initial states of the qubits and their final measurements are also created stored as Nodes.

5.3.3 Wire.h

The Wire.h header file contains the Wire class, which is a simple class. Each Wire stores pointers to the two Nodes it connects and its unique ID.

5.3.4 LineGraph.h

This header file contains the LineGraph class. The user creates an instance of a LineGraph class with a pointer to a Network as an input parameter if the user plans to contract the Network using line graph ordering. The only functions in the class other than the constructor run Quick-BB, a branch and bound algorithm for treewidth and tree decomposition, to determine the optimal line graph ordering, or contract the network based on a Quick-BB ordering.

5.3.5 ContractionTools.h

This header files contains the ContractionTools class. If the user wishes to contract their tensor network with the stochastic method provided, they should create an instance of this class. The stochastic method is preferred for small circuits where an optimal ordering is unnecessary, or if quickbb is not available. To create a ContractionTools instance, the user can either provide an already created network as a parameter or their QASM file path and measurement file path. In this way, the user does not have to interact with the Network class at all if they choose.

The class contains a contract method, where the user supplies which stochastic method they would like to use to contract the network. Of the non-LineGraph methods, we recommend exclusively using `Stochastic`. Here's a quick example.

```
#include "qtorch.hpp"
int main(){
qtorch::ContractionTools c("input.qasm","measure.txt");
c.Contract(Stochastic);
std::cout<<c.GetFinalVal();
}
```

It also has a separate contraction function for the user to contract the tensor network using a wire ordering of their choice. The other methods in the class help the user visualize the tensor network or calculate the treewidth. The user can print the tensor network to a graph file and then call another method within the class to calculate the treewidth or treewidth bounds of the tensor network. The final function in the class allows the user to retrieve the simulation output as a complex double when the contraction finishes.

5.4 Different Contraction Methods

There are multiple contraction methods the user can use to contract the tensor network, stochastic, user-defined, and line graph ordering. Here, we provide pseudo-code for the stochastic and line graph ordering methods.

Algorithm 1 Contraction via TD of L(G)

Create line graph L(G)
$\pi \leftarrow$ (Calculate optimal elimination ordering of L(G))
Eliminate wires of G in order π

Algorithm 2 Simple stochastic contraction

Define $G \leftarrow$ The tensor network Graph
$Threshold \leftarrow -1$
Define $MaxRejections$
repeat
    Choose a random wire w
    $(A, B) \leftarrow (Nodesofw)$
    $Cost \leftarrow rank(C) - max(rank(A), rank(B))$
    if Cost<=Threshold **then**
        Contract w to form node C
        $Rejections \leftarrow 0$
        $Threshold \leftarrow -1$
        Update G
    else
        $Rejections \leftarrow Rejections + 1$
        if $Rejections > MaxRejections$ **then**
            $Threshold \leftarrow Threshold + 1$
            $Rejections \leftarrow 0$
            Continue
until Graph completely contracted

5.5 Calculating Treewidth

Calculating the treewidth of a tensor network is important because the complexity of contracting a tensor network is $O(exp^{(treewidth)})$. Therefore, we provide a Quick-BB binary executable that does this. However, our qtorch executable provides a simple wrapper around the binary executable. Please see 5.1 for information on how to run Quick-BB using the linegraph method. The output file "output/qbb-stats.out" produced from running just Quick-BB will provide treewidth information on the underlying tensor network graph. To calculate the treewidth directly using the `quickbb_64` or `quickbb_32` binary executables[qui], the user must first create a CNF file that specifies their graph. The advanced format of a CNF graph can be found here: [cnf]. However, for a more basic explanation, see: [qui]. Once the CNF file is created, the user runs the binary executable provided using the format detailed here: [qui]. If the PATH environment variable was set, the user can type: `quickbb_64` or `quickbb_32` from any directory". Otherwise, run quickbb from the bin directory: `./bin/quickbb_64` or `./bin/quickbb_64` The output statistics file will provide a bound on the treewidth.

5.6 Other Executables Included

We include two other executables: maxcut.cpp and tests.cpp. If the user types the command: `make test`, it will compile and run the provided unit tests for the library. All test results will print to the "test.log" file in the output directory. The other executable, maxcut.cpp, is our implementation of QAOA (quantum approximation optimization algorithm) to solve instances of maxcut. We use our tensor network library to simulate QAOA and solve maxcut. To compile the maxcut executable, the user must use the command: "make cut". Please refer to the Examples section for how to run maxcut.

5.7 Extra Files

- Timer.h is a basic timer class where the user can access both wall and CPU time to time how long a simulation takes. Please see the actual header file for usage.
- preprocess.h is a file that we used for maxcut. The single method within it allows the user to determine a contraction sequence that runs faster than a provided maximum simulation time. It repeats stochastic method contractions that return after the maximum time or before the maximum time if a solution is found.
- GraphGenerator library: The graph generator is an extra library that allows the user to randomly generate X-regular graphs using the C++ standard library's Mersenne Twister algorithm. To generate one graph, the algorithm runs in $O(n * x)$, where n is the number of vertices in the graph, and x is the regularity of the graph. Within the library, there are two files: GraphNode.h and main.cpp. GraphNode.h is a simple struct that represents a single vertex in the graph being generated. The struct holds pointers to the other nodes in the graph the vertex is connected to and an integer that tracks the number of connections. The main.cpp file parses command line arguments provided by the user and runs the graph generation algorithm until all the graphs have been generated. At the start of the graph algorithm, n GraphNode objects are generated. Then, the algorithm attempts to place all of the $(n*x/2)$ edges though random selection of two vertices. If the random generation results in two already connected vertices or one vertex that already has x edges, the algorithm will try again, quitting after one hundred failed attempts. The edge placement loop completes regardless of whether all the edges have been placed successfully. If there is a mistake, the graph is discarded, and the algorithm is run again until success. A makefile is included in the library that allows the user to compile the executable: main.cpp. The command: `make all` is sufficient to do this. To run the executable, the user should type: `./main <reg> <numGraphs> <numNodes>` where the command line arguments: reg, numGraphs, and numNodes are respectively the regularity, the number of graphs to generate, and the number of vertices per graph. The user may find the generated graphs in the Output directory. Finally, if the user runs the executable twice without transferring the generated graphs to a separate directory, the graphs may be overwritten.
- maxcut.cpp: maxcut.cpp is an example of how tensor networks can be used to simulate QAOA to solve the MaxCut problem. To compile and run the maxcut executable, the user must

first install the nlopt (nonlinear optimization) library for C++, following the instructions in the folder included. Then, the user can run either the angle optimization (the first part of QAOA), the cut approximation calculator, or both. Type the command `make cut` to compile this executable. To run the angle optimization, the user must type the command:

```
maxcutQAOA <GraphFile Path> <QAOA p value> <0> <file path to output angle file>
```

if the shell PATH variable was set or

```
./bin/maxcutQAOA <GraphFile Path> <QAOA p value> <0> <file path to output angle file>
```

if the PATH variable was not set. To run the cut approximation calculator, the user must type the command:

```
maxcutQAOA <GraphFile Path> <QAOA p value> <1> <file path to input angle file>
<file path to output answer file> <seconds to preprocess for (optional)>
```

if the shell PATH variable was set or

```
./bin/maxcutQAOA <GraphFile Path> <QAOA p value> <1> <file path to input angle file>
<file path to output answer file> <seconds to preprocess for (optional)>
```

if the PATH variable was not set. Following the same pattern, to run both the angle optimization and cut approximator, the user must type

```
maxcutQAOA <GraphFile Path> <QAOA p value> <2>
<file path to output answer file> <preprocessing seconds (optional)>
```

if the shell PATH variable was set or

```
./bin/maxcutQAOA <GraphFile Path> <QAOA p value> <2>
<file path to output answer file> <preprocessing seconds (optional)>
```

if the PATH variable was not set. Note that running this executable can take a long time for input graphs with many vertices or high regularity. We also recommend a QAOA p value of 1. The p value from the angle optimization and the cut calculator must match, and the number of angles provided for the cut calculator must be sufficient.

6 Examples

There are multiple example scripts provided for the simple QASM circuit simulation, as well as sample measurement files, sample circuits, and the maxcut example. We invite you to explore these within the Samples and Examples folders of our library. Please type `chmod +x Examples/*` before running any of the example scripts.

7 Troubleshooting

If any bugs with the library are encountered, please contact the authors via email: *schuylerfried at gmail dot com, sawayanicolas at gmail dot com.*

8 Further Improvements

We want to make this library as easy to use as possible, so if you have any improvements you want to suggest, please send us an email. In the future, we hope to provide better parallelization using MPI as well as sparse tensors.

References

[Chu05] Isaac Chuang. Quantum circuit viewer: qasm2circ, 2005.

[cnf] Cnf file advanced description. http://www.cs.ubc.ca/~hoos/SATLIB/Benchmarks/SAT/satformat.ps.

[FGG14] E. Farhi, J. Goldstone, and S. Gutmann. A Quantum Approximate Optimization Algorithm. *ArXiv e-prints*, November 2014.

[qui] Quick bb binary. http://www.hlt.utdallas.edu/~vgogate/quickbb.html.