# Script – Thesis’s defence presentation:

“Normal” slides:

Slide 1: Cover slide

(Total: 3947 words. Without correlation-based iQAQE: 3753 words. Check the time on this! Most recent: 3499 words. Most, most recent: 3022 words. Final update: 2967 words.)

Good morning, everyone, today I’m going to be talking about my work on “Novel Variational Quantum Algorithms for MaxCut”.

Slide 2:\* Index

First, I’ll look to explain what the title means, more in-depth. This will allow me to introduce the problem that we’re interested in solving, as well as, quantum computing, variational quantum algorithms, and the iQAQE Framework, that we’ve developed. After that, I shall present the results of our investigations, followed by some conclusions and thoughts regarding possible future work.

Slide 3: “MaxCut” (title)

Starting off, let me introduce the MaxCut problem.

Slide 4:\* MaxCut problem

MaxCut stands for “Maximum Cut” and is a fundamental problem in graph theory and combinatorial optimization, which involves partitioning a graph G into two disjoint subsets, such that the number of edges connecting nodes of different sets is maximized. Formally, this can be expressed by the maximization of this quantity, the “Cut”.

MaxCut finds a number of use cases in areas like machine learning, statistical physics, circuit design and data clustering.

There is also significant interest from the computer science community in terms of computational complexity. MaxCut is known to be NP-hard, which means that one can formulate any other NP problem, such as these three, as a MaxCut problem. Thus, finding an efficient solution to MaxCut results in an efficient solution to any other NP problem, further motivating our work.

Slide 5: MaxCut – state-of-the-art

Now, a number of algorithms have already been proposed to solve the MaxCut problem in an approximate manner. Classically, the Goemans-Williamson algorithm stands out, which boasts the best-known guaranteed approximation ratio to date, of 0,87856. This is the ratio between the worst performing instance of the algorithm and the MaxCut value. If this number were 1, the algorithm would be perfect.

On the other hand, we also have hybrid quantum-classical algorithms, such as the Quantum Approximate Optimization Algorithm, or QAOA in short, and the Qubit-Efficient MaxCut Heuristics Algorithm, or QEMC. Our work will consist of an interpolation between these two, aiming to amalgamate their strengths into a better-performing algorithm.

Slide 6: “Quantum” (title)

Next up is the “Quantum” part.

Slide 7: Quantum computing – general concepts

And we’re talking about quantum computing. This is based on exploiting the principles of quantum mechanics to perform calculations, using the quantum analogue of classical bits, qubits. A single qubit can be represented as can be seen here, in a superposition of its two basis states, \ket0> and \ket1>. Alternatively, a multi-qubit state can be represented as a linear combination of all its associated basis states, expressed as the tensor product of each individual qubit’s basis states, like this. Besides superposition, quantum computing also exploits entanglement as a means to achieve computational speed-ups, although it is yet unclear exactly how this comes about.

Slide 8: Quantum computing – types of qubits

As a small sidenote, I’d like to briefly mention the types of physical systems that can be used as qubits for quantum computing. More commonly, we see qubits based on either photons, neutral atoms, trapped ions, semiconductors, such as quantum dots, and superconducting circuits.

Slide 9:\* Quantum computing – quantum circuits

**Currently**, most quantum computers are being designed with a circuit-centric architecture in mind. A quantum circuit can be thought of as the quantum analogue of traditional logic circuits. Notably, we can have single, or multi-qubit quantum gates.

The state of a single qubit, living in a 2-dimensional Hilbert space, can be represented by a vector in the Bloch sphere, which you can see here. Some of the most commonly used quantum gates are rotations around each of these three Cartesian axes and are given by complex exponentials of the well-known Pauli matrices. Here you have them.

As you can see, all of these quantum gates can be represented mathematically by a matrix, which elucidates how they act on the qubits. This also means that it is entirely possible to simulate a quantum circuit using classical computers. The catch, however, is that the number of basis states that can be encoded with n qubits is 2^n, an exponential scaling, which makes it impossible to simulate more than a few dozen qubits in practice, before we run out of memory.

Slide 10: “Variational Quantum Algorithms” (title)

At this point, before we proceed to our original work, I need to explain what Variational Quantum Algorithms are.

Slide 11: Hybrid Quantum-Classical Computing

These are the hallmark of Hybrid Quantum-Classical Computing, which combines both classical and quantum computing paradigms to leverage the strengths of each. Generally, a parameterized quantum circuit works in tandem with a classical optimizer to solve complex problems. This cooperative approach allows for mitigating the challenges associated with quantum computing, while maintaining quantum circuit depths shallow. The depth of a quantum circuit is related to the number of gates that are sequentially applied. Each gate introduces some amount of noise into the system, such that it is always better to have shallower circuits, whenever possible. Besides this, a number of prospective applications have already been proposed for these Variational Quantum Algorithms, or VQAs, in short, such as: […].

Slide 12: Variational Quantum Algorithms – applications

[…] combinatorial optimization, machine learning, doing quantum simulations, solving systems of equations, and many more, as you can see in this figure.

Slide 13:\* VQAs – basic structure

Now, I should also introduce the basic structure of a VQA, as this will be relevant to my work. First, understand that a VQA merely solves the minimization problem of a certain cost function. Often times, the cost will have this functional form, which corresponds, essentially, to a sum of expectation values.

Besides this, a VQA also features an ansatz. This refers to what I previously called a parameterized quantum circuit. Moreover, there’s a difference between what we call problem-inspired and problem-agnostic ansätze, which stems from how much of the specific problem’s information we include in the ansatz’s design. Generally, it is always better to use problem-inspired ansätze, but they come with some design challenges.

Lastly, it’s **important** that we’re able to efficiently estimate gradients, as they’re **important** for the classical optimization procedure. This is usually done by employing the so-called parameter-shift rule, which you can see here.

The training then proceeds in what is called a “hybrid loop”, where we create the quantum state, using the parameterized quantum circuit, followed by the estimation of the cost. This is, then, used by the classical optimizer to update the ansatz’s parameters, in the direction of the minimum cost, entirely similar to how one would train a neural network, for example.

Slide 14:\* Quantum Approximate Optimization Algorithm (QAOA)

The next thing I’d like to mention are the two algorithms which inspire our work. This is the first one, called the QAOA. In QAOA, we replace the usual classical variables with Pauli-Z operators, which are used to build what is know as the problem or cost Hamiltonian, here. This is defined to be akin to the usual “cut”, that we saw before. The cost (function) itself is, then, defined as the expectation value of this cost Hamiltonian.

Furthermore, the QAOA ansatz features alternating blocks of parameterized time evolution according to either the cost or mixer Hamiltonians. The role of this mixer Hamiltonian can be understood as introducing fluctuations, helping the algorithm explore the solution space more effectively.

Note that QAOA uses a number of qubits equal to the number of graph nodes. This allows for a very direct encoding of the graph’s partition in the most frequently sampled basis state, after training. Simply, if qubit 0 is \ket0>, node 0 belongs to set 0; if qubit 0 were \ket1>, then it’d belong to set 1.

Slide 15: Qubit-Efficient MaxCut Heuristic Algorithm (QEMC)

Lastly, there’s this QEMC algorithm, which has a number of crucial differences from QAOA. First, it features a probability threshold encoding scheme. Now, we don’t have one qubit per graph node, but one basis state per graph node. This allows for an exponential reduction in the number of qubits. Also, through this encoding scheme, we just look at each node’s associated basis state’s probability to decide on that node’s color: if the probability is above a certain threshold, it belongs to set 1, otherwise it is in set 0.

To complement this encoding scheme, QEMC uses a different cost function, which can be seen here, in which the graph’s structure is implicitly encoded.

To top everything off, QEMC makes use of a problem-agnostic ansatz, called the “Strongly Entangling Layers” ansatz, which can be seen here. This ansatz features single-qubit parameterized rotations, interspersed with entangling CNOT gates.

Slide 16: “Novel” (title)

Finally, we are ready for our original work. [Time: 9:43.]

Slide 17:\* iQAQE Framework

We propose something called the iQAQE Framework. The name iQAQE stands for Interpolated QAOA/QEMC. It builds on QEMC’s core components while integrating concepts from QAOA to leverage the strengths of both algorithms. Differently from QEMC, though, it associates a list of basis states to each graph node, instead of just one.

Additionally, it allows for a number of parameters, which one can change to generate entirely different VQAs, hence why we call this a framework. More specifically, the number of basis states which we decide to associate with each node can vary in between 1, which is QEMC’s limit, and 2^{n-1}, which is QAOA’s limit. We call this the “list cardinality”. Also, the number of qubits, too, sits in between QEMC’s and QAOA’s limits.

Implementation-wise, iQAQE makes use of QEMC’s encoding, cost function and ansatz, but adapted to the correct number of qubits. This requires us to transform the basis states’ probabilities into node probabilities, so that we can use QEMC’s cost. In practice, this is done, simply, by adding the probabilities of each node’s associated basis states and normalizing the results so they add up to 1.

Besides the previously mentioned parameters, there’s also what we call the “mapping”, which refers to both the process of distributing basis states over the lists, as well as a specific allocation of basis states among the lists. Additionally, there’s a number of hyperparameters that can be tuned, such as the number of ansatz layers and number of shots. What I refer to as shots is how many times we sample the quantum circuit’s output, for computing the relevant quantities for the cost, be it expectation values, or probabilities.

There’s a number of advantages one can postulate for such a hybrid scheme. More obviously, the fact that it uses less qubits and shallower circuit depths than QAOA. Also, it should require less shots than QEMC, in practice. And, finally, its tunable “quantum-ness”, being an interpolation, could prove to be a valuable asset.

Slide 18:\* Implementation

Implementation-wise, all our classical simulations of quantum circuits were run on my laptop, using the PennyLane and NetworkX Python libraries.

I’d also like to point out the performance metrics which are used to evaluate our algorithms and schemes. These can be training curves showing the evolution of the cost or approximation ratio during training. Or, alternatively, some more statistically robust metrics, such as average training curves, out of 5 or 10 runs, with different random initial parameters. Or, even, average/median best-so-far curves. This best-so-far refers to a transformation that I apply to the training curves, such that only the best-so-far cut is maintained. [Time: 13:05.]

Slide 19: iQAQE schemes & results

We’re finally ready to present some results, from the schemes that we’ve devised during this work. There’s actually quite a large number of developed heuristics which you can see here, but since we don’t have time to go over all of them, let’s focus on thesefive.

Slide 20: Problem graph

Before anything else, all of the results that I’ll be presenting here respect this simple 8-node graph.

Slide 21: Random iQAQE

The first test that we did was entirely random, in that we chose the number of qubits, list cardinality and “mapping” all at random, and ran some simulations, for both finite and infinite shot numbers. With this, we just wanted to assess the general performance of the scheme, particularly how it varies based on the number of shots.

And these were the results: here, for infinite, 1024, 512 and 256 shots. Note the y-axis, where we plot the best-so-far average approximation ratio, as previously discussed. The first thing that ­stands out is how QEMC’s results appear to deteriorate as the number of shots decreases. This is expected, as QEMC depends on the estimation of the circuit’s output’s entire probability distribution, which requires more shots to be statistically accurate, than, say, expectation values, used in QAOA. Surprisingly, however, iQAQE seems to exhibit slightly greater resilience to this effect, despite sharing QEMC’s cost function and ansatz. This resilience might be attributed to the presence of multiple basis states associated with each graph node, potentially reducing the need for exhaustive sampling.

Slide 22: Basic polynomial compression-type iQAQE

Now, we propose some more concrete heuristics for doing the “mapping”. This first scheme is called “Basic Polynomial Compression-type iQAQE” and, as the name suggests, allows for a polynomial compression in the number of required qubits, compared to QAOA. In practice, for each graph node, we fix k different qubits to 1, and associate those basis states to that node. The easiest way to understand this is by looking at an example. For instance, check this “mapping”, using 5 qubits, with k=2. The number of nodes that we can encode is given by “5 choose 2”, here. As this is greater than 8, we can use this for our 8-node graph.

Here are the results. Note that this time, we use the average best-so-far approximation ratio. As can be seen, again, QAOA tops the charts, however, we do have something competitive with QEMC, even surpassing it, for k=2. The biggest advantage of this scheme is, exactly, the polynomial reduction in the number of required qubits.

Slide 23: Correlation-based iQAQE (Remove?)

Next up is the “Correlation-based iQAQE” scheme. This is analogous to what we’ve just presented before, but also allows for 0s to be fixed. The scheme aims to identify correlations among the different qubits, wherein ‘correlations’ denote identical colors. And here are the results. This time, we use the average approximation ratio, which we’ll mostly stick to from now onwards. Also, these shaded regions represent the minimum and maximum approximation ratios achieved out of the 10 training curves, at each iteration.

Back to the results, we notice a sharp underperformance. This is likely due to the large overlap of 50% between every two pairs of lists. We believe this makes it harder to tune one node’s color without significantly affecting other nodes’ colors, making it harder to optimize the scheme.

Slide 24: Unmodified extended-QEMC

Next up, we’ve built another heuristic for doing the “mapping”, based on an extension to the regular QEMC model. This “Extended-QEMC” scheme is, essentially, QEMC, but with m extra qubits added. The motivation behind this has to do with the fact that the original QEMC scheme is easily classically “simulable”, due to the exponential compression in qubit number. By adding some more qubits, we’re able to fight off this classical simulability.

The ”mapping” is done similarly to what we do in QEMC, but, this time, by associating 2^m basis states to each list, instead of just 1. Note that we still use all basis states with no overlap between lists.

Here are the results for this scheme. Again, it is competitive with QEMC, even surpassing it at times, but still not quite on par with QAOA.

Slide 25: Alternative ansätze

We also utilized a more problem-informed ansatz, which consisted of including an R\_x gate, before each CNOT gate. This allows for dynamically controlling the amount of entanglement introduced in the system. The results are shown here. We can achieve slight improvements by introducing this modification to the ansatz, which is a nice result, in and of itself.

Slide 26: Goemans-Williamson & bigger graphs (32-node graph)

The final thing I’d like to present is a small scalability analysis to larger **graph** instances and a comparison against the classical state-of-the-art in the Goemans-Williamson algorithm. Hereare the results for a 32-node graph.We present QEMC, with and without the ND-CNOTs that I was just describing. Also, Extended-QEMC, with just a single basis state selected, is presented, for many combinations of m and number of ansatz layers. For the first time in our analysis, we have surpassed the classical state-of-the-art with one of our heuristics, as can be seen by this pink line, which goes over the white line and its associated shaded region, which is a substantial result.

Slide 27: Goemans-Williamson & bigger graphs (100-node graph)

However, if we now look at **the** 100-node graph, this is no longer the case. We suspect that this might have to do with the additional complexity of the optimization landscape as more qubits get added to the system. Nevertheless, now, the Extended-QEMC scheme performs better, reaching a level competitive with the classical state-of-the-art, which is an inspiring result.

Slide 28:\* Conclusions and future work

There’s a number of things that we can take away from the iQAQE Framework. First, there’s some advantages, like the reduced qubit requirements and the enhanced resilience to finite shot noise. In addition to that, the fact that it constitutes a framework allows for the exploration of many uncharted VQAs. Regardless, we’ve verified performance competitive with the classical state-of-the-art, which is encouraging.

Now, there are a number of things that can be pursued in future work. Namely, there’s something I didn’t have time to go over in this presentation, but we’ve also proposed a small machine-learning model for attempting to predict which “mapping(s)” will perform better, for a certain graph. This model’s performance was somewhat sub-par, but we believe it can still be improved.

We’ve also proposed alternative schemes for solving MaxCut. These are the parity-like QAOA and the batch-based Oracle coloring. These, however, have some limitations, which we’ve identified, that hinder their performance. Nevertheless, these could still be further explored.

Finally, there’s still work to be done in the framework itself. This could range from trying more problem-informed ansätze, to finding better “mapping” heuristics. Another thing I would’ve liked to have done is a more significant scalability analysis, for larger graphs. I’m talking about a few thousand nodes, as this would allow for a more significant comprehension of these schemes’ scalability. [Time: 21:15.]

Slide 29: Acknowledgements

At this point, I’d like to thank my research supervisor Yasser Omar, alongside my colleagues Zoltán Zimborás, Miguel Murça and Bence Bakó, which continuously supported me throughout this work, providing me invaluable guidance. Thank you.

Slide 30: Main bibliography

And this is my main bibliography, in case you’re interested in anything in particular. And that is all, thank you so much for your attention. [Time: 21:35.] [Most recent time: 19:52.] [Most, most recent (final): 19:05.]

Back-up slides:

Slide 31: Fixed-parity iQAQE (Remove\*)

The last polynomial compression-based scheme that I present is the “Fixed-parity iQAQE”. This time, we fix the parity of the k selected qubits to be even. The parity of a basis state refers, essentially, to the number of 1s in said state. If that number is even, the parity is even, otherwise, it is odd. This scheme was proposed as a heuristic alternative to the previous correlation-based iQAQE. As an example “mapping”, this is what we use for our 8-node graph.

And here are the results. This time, we see performances on par with QEMC, but never quite surpassing it. And, again, QAOA outperforms both methods significantly, for this 8-node graph. We also note that k=2 performs especially worse. This case actually corresponds exactly to the previous correlation-based iQAQE scheme, for k=2, as requiringtwo qubits to be even is equivalent to havingthem be “correlated”, i.e., \ket00> or \ket11>. We suspect that the underperformance stems from the same reason as before, for this case.

Slide 32:

Write something here.

Slide 33:

Write something here.

Slide 34:

Write something here.

Slide 35:

Write something here.

Slide 36:

Write something here.

Slide 37:

Write something here.

Slide 38:

Write something here.

Slide 39:

Write something here.

Slide 40:

Write something here.

Slide 41:

Write something here.

Slide 42:

Write something here.

Slide 43:

Write something here.

Slide 44:

Write something here.

On the various metrics used:

Whereas the “average best-so-far” gives information about the top performance of this algorithm, on average, the simple, bland, “average” gives information about the average performance, more susceptible to fluctuations in the training, constituting a richer metric. Of course, in practice, we’d use the best-so-far, as that’s what we’re interested in.