# 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!)

Good morning, everyone, today I’m going to be talking about my thesis, entitled “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, MaxCut, 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)

First 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”, where this Chi is 1, when the two input nodes belong to different sets, and 0 otherwise.

MaxCut finds a number of use cases in areas like machine learning, statistical physics, circuit design and data clustering, as it can be used in binary classification tasks, simulation of spin-glass models and VLSI, Very Large-Scale Integration, design, just to name a few.

There is also significant interest from the computer science community in terms of computational complexity. **MaxCut is known to be NP-hard, which makes it a significant computational challenge to efficiently find an optimal solution to the problem. In any case, it being NP-hard means that one can formulate any other NP problem, such as these three, as a MaxCut problem. In turn, this means that 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, two stand out: the Goemans-Williamson algorithm, 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, cut-wise, and the MaxCut value; Additionally, the Trevisan algorithm, with Soto’s refined analysis stands as the state-of-the-art spectral method, based on eigenvalues and eigenvectors of the graph’s adjacency matrix.

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 of sorts between these two, aiming to amalgamate their strengths into a more robust, and better-performing, algorithm.

Slide 6: “Quantum” (title)

Next up is the “Quantum” part. What exactly is “Quantum” here?

Slide 7: Quantum computing – general concepts

Here, we’re talking about quantum computing, of course! 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, in other words, a superposition, of all its associated basis states, where these basis states are 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 under certain scenarios, although it is yet unclear exactly how this comes about. Entanglement refers to correlations between the qubits, with no equivalent classical counterpart.

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, just to name a few.

Slide 9:\* Quantum computing – quantum circuits

Now, most quantum computers are being designed with a circuit-centric architecture in mind. A quantum computer is no more than a quantum circuit, which can be thought of as the quantum analogue of traditional logical circuits, based on logical gates, such as AND, OR, NOT, etc. We can have single, or multi-qubit gates.

The state of a single qubit, living in a 2-dimensional Hilbert space, can be represented in what has become known as the Bloch sphere, which you can see here. A vector on the sphere’s surface represents a specific state the qubit can be in. In this vision, single-qubit gates will correspond to some transformation which rotates the initial state vector in this Bloch sphere to some other place in it. Speaking of rotations, some of the most commonly used quantum gates are rotations around each of the three Cartesian axes, 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. This elucidates how they act on the qubits, which are vectors in C^2. This also means that it is entirely possible to simulate a quantum circuit using classical computers, by sequentially applying each gate’s corresponding matrix representation to the input quantum state. 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 hallmarks 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 more efficiently **than either could achieve alone**. This cooperative approach allows for mitigating the challenges associated with quantum computing, while maintaining quantum circuit depths shallow, which further mitigates the noise. For reference, the depth of a quantum circuit is related to the number of gates that are sequentially applied, **in said circuit**. 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 **theorized or 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 following optimization problem: minimizing the value of a certain cost function, which depends on parameters \theta. Often times, the cost will have this functional form, which corresponds, essentially, to a sum of expectation values, of some operators, O\_k.

Besides this, a VQA also features an ansatz. This refers to what I previously called a parameterized quantum circuit. As the name suggests, it is merely a quantum circuit, which contains some parameterized quantum gates, such as the previous rotation gates, that I showed. 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 estimate gradients efficiently, 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, using the final ansatz state. 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, **in classical machine learning.**

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, that I’ve already briefly mentioned, called the QAOA. In QAOA, we replace the usual classical variables with Pauli spin ½ operators, Z\_i. These are just Pauli-Z operators and are used to build what is know as the problem or cost Hamiltonian, here. This is supposed to be akin to the classical cut, that we saw before. The cost (function) itself is defined as the expectation value of this cost Hamiltonian.

**Now that we’ve specified the cost**, we’re only left to specify the ansatz, to fully characterize this VQA. The QAOA ansatz is called the Quantum Alternating Operator ansatz, **unsurprisingly**. It features alternating blocks of parameterized time evolution according to either the cost or mixer Hamiltonians. This mixer Hamiltonian is given by the sum of Pauli-X operators over all the N qubits, for N graph nodes. The role of this mixer Hamiltonian can be understood as introducing fluctuations, helping the algorithm explore the solution space more effectively, ideally preventing it from getting trapped in sub-optimal local minima.

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. It has a number of crucial differences from QAOA. First, it features a novel 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. 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**, for three qubits and two layers. The layers refer to these repeating blocks that **can be seen here**, using different parameters for each block. This ansatz features single-qubit parameterized rotations, interspersed with entangling CNOT gates, to generate a highly entangled quantum state.

Slide 16: “Novel” (title)

Finally, we are ready for our original work. Now, what is “Novel” in what we do?

Slide 17:\* iQAQE Framework

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

As such, it allows for a number of parameters, which one can choose to generate entirely different VQAs, hence why we call this a framework for designing new VQAs. 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. 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 up the probabilities of each node’s associated basis states and normalizing these node probabilities so they add to 1.

As we’ve mentioned, there’s a number of parameters to be chosen/tuned to generate different VQAs, namely: the qubit number, the list cardinality (this is just how many basis states go into each list), and the “mapping”. This “mapping” 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, as well as Adam’s learning rate, as this is the classical optimizer that we decided to use. 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, in QAOA, or probability distributions, in QEMC and iQAQE.

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 Python library, for differential programming of quantum computers, alongside NetworkX, for graph manipulation and treatment.

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/approximation ratio is maintained. More frequently, though, we shall simply use the average training curve.

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 a quite large number of heuristics developed which you can see here. Since we don’t have time to go over all of them, let’s focus on these six.

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. (**Remember, AR = 1 is the best-case scenario!**) The first thing that ­stands out is how QEMC’s results appear to significantly deteriorate as the number of shots decreases. This is expected, as QEMC depends on the estimation of the quantum 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. This is the first meaningful conclusion that we drew from iQAQE’s simulation.

Slide 22: Basic polynomial compression-type iQAQE

Now, we propose some more concrete heuristics for doing the “mapping” of the basis states to the lists. 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 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 the “Combinations of 5, 2 by 2”. This is just the number of ways through which we can choose two qubits to fix to 1, from the available 5. As this is greater than 8, we can use this for our 8-node graph. And that’s what we did, just using the first 8 lists, since we don’t need all 10.

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 now appear to 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. This approach was inspired by the notion that by identifying correlations between nodes, it becomes feasible to color the entire graph as long as one node is initially colored.

And here are the results. This time, we use the average approximation ratio, which we’ll mostly stick to from now onwards. We’ve been using other metrics to illustrate how different information can be retrieved **by utilizing these distinct metrics**. But for studying the algorithm, this bland “average” is also quite relevant. 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 hard to optimize the scheme.

Slide 24: Fixed-parity iQAQE

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 25: 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. This way, we still use all basis states, with no overlap between lists! In practice, we attribute the first 2^m to the first node, the second 2^m to the second, and etc.

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 26: Alternative ansätze

I don’t have time to go over this one in detail, but I didn’t want to leave **it** out, as I reckon it’s quite relevant, but let me just give you some words, here. We also utilized a more problem-inspired ansatz, which consisted of including an R\_x gate, before each CNOT gate, in the “Strongly Entangling Layers” ansatz. This is supposed to allow for dynamically controlling the amount of entanglement introduced in the system, by enabling the CNOTs’ control qubits to rotate in and out of the state which activates the CNOT gate. The results are shown here. It appears that we can achieve slight improvements by introducing this modification to the ansatz, which is a nice result, in and of itself. **(Twenty minutes, here!)**

Slide 27: 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, that’s what we call them. Also, Extended-QEMC, with just a single basis state selected, at random, is presented, for many combinations of m and number of ansatz layers. For the first time in our analysis, it appears that 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. Indeed, ND-CNOT QEMC’s maxima go over the maxima attained by the G-W algorithm, which is a substantial result.

Slide 28: 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. Forcing the introduction of ND-CNOTs is forcing the introduction of O(n) parameters in the problem, for n graph nodes. Nevertheless, it appears that, now, the Extended-QEMC scheme performs better, reaching a level competitive with the classical state-of-the-art, which is an inspiring result.

Slide 29:\* Conclusions and future work

Finally, we’ve reached the end. 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, relative to QAOA, 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, which might perform better, or worse, than the heuristics that we’ve presented here. Regardless, we’ve verified performance competitive with the classical state-of-the-art, even for larger graphs, which is inspiring, especially for future work.

Now, there are a number of things that we’ve yet to do here, or that can be expanded on, to perfect them. 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, constituting an avenue for future research.

In addition to that, we’ve also explored alternative schemes for solving MaxCut, not iQAQE-based, but stemming from our investigations on iQAQE. These are the parity-like QAOA and this batch-based Oracle coloring. These, however, have some limitations, which we’ve identified, that hinder their performance. Nevertheless, these could still be further explored, in the hopes of finding workarounds for such limitations.

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. Perfecting the previous heuristics can also be done, by performing more complete grid-searches over the model’s hyperparameters, for example. 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, which we could then compare with the state-of-the-art, in a more meaningful setting.

Slide 30: 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 31: 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.

Back-up slides:

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.