Using the Quantum Approximate Optimization Algorithm (QAOA) to Solve Binary-Variable Optimization Problems

Featuring Jason Larkin and Daniel Justice

Welcome to the SEI Podcast Series, a production of the Carnegie Mellon University Software Engineering Institute. The SEI is a federally funded research and development center sponsored by the U.S. Department of Defense. A transcript of today's podcast is posted on the SEI website at sei.cmu.edu/podcasts.

Dan Justice: Hi, and welcome to the SEI Podcast Series. My name is Dan Justice, and I am a software developer in the SEI's AI [Artificial Intelligence] Division. Today, I am joined by my colleague Dr. Jason Larkin here, a senior researcher who also works in the AI Division. We are here to discuss a recent paper that we coauthored on the Quantum Approximate Optimization Algorithm, also known as QAOA. It is a hybrid quantum classical algorithm that is used to solve binary variable optimization problems. Jason and I have both been guests on this podcast before to talk about our previous work in the field of quantum computing. We are here again to elucidate more on what we have done and what we are going to continue to do. For those in the audience who have not seen our previous podcast, there will be links down below. We are going to start off by talking a little bit about ourselves and reintroduce ourselves and what we do here at the SEI.

So, Jason, how's it going? What do you say we do here?

Jason Larkin: Oh, well, we work on all kinds of things in quantum computing, up and down the quantum computing software stack, from the applications and algorithms level to compilation, all the way down to the quantum bare metal so to speak.

Dan: Are we doing any specific algorithm design or hardware design here at all?

Jason: No. I think we are like kind of agnostic to those things. We mostly respond to the different architectural designs. That is, we have to incorporate them into the software design and the compilation process. But we ourselves don't do that, we let the companies and the people building them do that. But we have to be cognizant of it. We have to include those effects in our work.

Dan: Right. Yes. There is a huge amount of software code that we find ourselves doing. But we are not exactly driving each of those individuals themselves. So our previous work that we talked about was the beginnings of our looking into the NISQ [noisy intermediate-scale quantum] era, the beginnings of our research into this QAOA algorithm and what we are looking at. So, this paper. What is your paper? Where is it going to be published, and what is it about?

Jason: The title is Evaluation of QAOA Based on the Approximation Ratio of Individual Samples. It is going to be published in the journal *Quantum Science and Technology*. It is available now in preprint. It will officially be published soon.

Dan: It is officially accept, correct?

Jason: It is officially accepted.

Dan: Congratulations.

Jason: Thank you. We could sort of unpack what all that means. So first, we are evaluating this QAOA algorithm for whether or not it can be performant against the alternatives to solving a particular problem. So, in the paper, we pick this problem MaxCut, and we want to figure out, does this QAOA algorithm have a chance to compete against alternatives of solving this problem? All these alternatives being classical solvers on classical computers.

Dan: Real quick though. What is MaxCut, and why would, say, any individual in the world be interested in this?

Jason: So MaxCut is a problem of if I give you a graph, the point is to label the nodes on the graph such that there is a maximal number of disagreements between edges that connect these nodes, and this actually cuts the graph. So, the problem MaxCut is to find the largest cut of this graph. For practical purposes, this MaxCut problem shows up in a number of domains. One interesting one is in like VLSI [very-large-scale integration design]. You design chips. This MaxCut problem is, I forget exactly what it does, but it's important for that problem; and there are other practical problems. Then this MaxCut problem is also related to other NP-hard problems via this list called Parch 21. This goes all the way back to like the '70s. So this MaxCut problem is related to those. I think that one of the reasons we picked this problem is because not only does it have practical relevance, it also has a lot of classical existing benchmarks for the problem. Then I think some of things we learn about this particular problem we can also translate to these other types of graph problems. So we learn a lot in general about solving those kinds of problems. But we also just wanted to look at one problem, because the scope of the paper is large.

Dan: Jumping ahead, did you find that this quantum classical hybrid algorithm, QAOA, did you find that it was more efficient or outperformed the classical or not?

Jason: What we found was, if you take this QAOA algorithm, and you execute it in sort of a vanilla kind of way, then you find that it is actually not that performant. We found an improvement in the performance of this algorithm if we do what we called a training procedure. So what that basically means is that we train this QAOA algorithm. In the algorithm, it's a variational algorithm, like many of the other NISQ algorithms. In this algorithm, there are variational parameters. The difference in performance of solving a particular problem, like MaxCut, with random parameters versus optimized parameters is very, very large, like orders of magnitude difference in performance. So, the key becomes, *How do I find these optimized parameters*? So the other issue is that what if you had to find these optimal parameters for every system size as the system grows larger. The search base will grow larger, and the overhead associated with it will grow larger. But what we found is that you could train the algorithm on instances with a problem of small size and then use those same trained parameters on larger instances, thus eliminating a significant amount of overhead overall in the algorithm. If you do all of that, and then in our paper, we were only able to do noiseless simulations, so that is a major caveat to all this. If you do all that, then you can make this QAOA performant at least with the classical solvers on the classical hardware that we compared to in our paper.

Dan: OK.

Jason: It is by no means optimal, but it is about as optimal as you can get with commercial, off-the-shelf sort of hardware and algorithms. But the major caveat is that this all worked in a noiseless simulation. If one applies noise, as most all of the devices we have in NISQ era are noisy, then the performance that we predicted is going to decrease. So the real question becomes, *Can you utilize the procedure that we did in the paper to scale and also tolerate this level of noise while maintaining performance?* You can think of what we did in the paper as like an upper bound on the performance to some extent. What we need is, we need significant improvements in hardware, even within the NISQ era. But there are interesting results occurring, especially within the last year. The jury is still out. I think at the end of the day, we can't just simulate these devices, we are going to need to get actual benchmarks on real machines. Pretty soon, we are going to have machines with numbers of qubits that we won't be able to simulate classically.

Dan: Right. About how many qubits you think that is going to be?

Jason: Well, that all depends on the type of circuit. But, generically speaking, you can say, *Once you get around 50 qubits, circuits with 50 qubits, then it starts to become hard to simulate them classically*. This is not true for all circuits. There are a lot of circuits that can be simplified significantly. Even for this QAOA algorithm, for example, there are certain graph types where, if you make simplifications of the circuit, one can simulate circuits with 100 or 1,000 qubits. But in general, you are going to start hitting classical intractability around 50 qubits, certainly by 100 qubits. So yes, we are just sort of right in the era of having machines of that size. What is more critical though even than that may be the circuit depth. We need machines that can run operations for much deeper circuits than we have available right now.

Dan: OK, yes. About how many qubits and how many gates were you utilizing in your experiments?

Jason: I think the biggest system we did in the paper was 32 qubits, and the depth of the circuits depends on the graph type, how connected the graph is. I think the sort of minimum circuit depths, by the time you get to those sizes, are probably in the hundreds of two qubit gates. Then for complete graphs, it should be something like the square of the number of nodes, so like maybe 1,000 gates. We don't have machines that can reliably execute circuits of those depths. Although maybe IQM is getting close to that. That is the challenge. We really need to expand or rather to improve the hardware, to at least be able to execute circuits of those widths and depths.

Dan: At this point, most quantum computers, if you have that kind of depth, you are basically creating a random noise generator.

Jason: Or even worse than that, I think.

Dan: You think so?

Jason: Yes. It's just noise.

Dan: Just noise.

Jason: Yes.

Dan: I am curious though, why is it so difficult to simulate the noise of a quantum machine?

Jason: Oh, because basically there are a number of ways of simulating noise. But the sort of most straightforward way to do it is you essentially have to take your noiseless circuit and then you have to apply some noise model. Like you say, *All of my one-qubit gates are going to have an error rate of, you know, one in every 100 gates or one every 1,000*. You do this for all gate types. Then, to actually simulate noise, you need to take that noiseless circuit and insert gates randomly according to the probability of that error occurring. Then you also need to average over different realizations of those insertions of errors. Roughly speaking, whatever it cost to simulate noiseless, you are going to need to add an extra overhead of 100 to 1,000 to do the noisy case. It is about 1,000 times more computationally costly than doing noiseless.

Dan: OK, and if we are already running up on the limits of your laptop, or say even a supercomputer, increasing it a thousandfold is just not tractable?

Jason: Yes, and then another reason would be that we have a pretty good idea of how the noise is going to affect our results. Namely, it is going to decrease the results. So again, you can sort of look or think about the work that we did as kind of an upper bound on the performance. It only gets worse from here with noise. It is for two reasons. That reason, and also, we didn't have enough computational resources to do it.

Dan: OK. Now, the title of the paper is Evaluation of QAOA. On what criteria did you evaluate it, and against what classical algorithms did you compare it to?

Jason: Basically, the way we evaluated the algorithm was the same way you would evaluate any algorithm on any kind of hardware, whether it's classical or even neuromorphic or any kind. You are basically looking at the times of solution. So how long does it take to get a solution of a certain quality? Now, you could say, *Well, I want the solution that is optimal. I want the very best solution, the maximum cut*. Well, this problem is NP hard. So in general, waiting around for the best solution is going to take exponential time. So that would be an exact solution, so we compared the results. For QAOA, we actually looked across a range of the solution qualities, let's say from 0.8 to 1, where 1 equals the best and 0.8 is 80 percent of the best. We evaluated QAOA between those limits, and then we compared those results against classical exact solver, so one that will find the absolute optimal cut but going to take exponential time in general. Then we also compared against approximate solvers. An approximate solver is one that gives a guarantee on a minimum quality of results in a certain time. A famous example of this is the Goemans-Williamson algorithm, which is based on semi-definite relaxation, and this algorithm is approximate because it has a quality guarantee, and it does so in polynomial time.

Dan: Right.

Jason: Then you have what are called heuristic solvers. These solvers in general don't give you any guarantee on performance. You just have to try them out and see what they give you. We compared all of these things to QAOA. Also, for QAOA, there are only a limited number of known performance results. In general, QAOA is very heuristic. You just have to try it out and see how it performs. So we basically adopted that mindset. We treated basically QAOA as a heuristic and then examined what its performance was for a range of qualities that are both approximate and also exact.

Dan: Yes, that is awesome. Are you expecting others to adopt these new metrics for the QAOA algorithm?

Jason: I am not sure if I am expecting them to. I think more or less the people doing this kind of work the most vigorously were already doing something similar to what we suggested. That being said, we didn't see in the literature a very sort of cohesive and single study that utilized all of the things we did in the way we did, which was kind of odd. Do I expect people to adopt? I think so. I think if you are doing this kind of benchmarking rigorously, I think you sort of have to do what we did, unless you can figure out better analytical results for QAOA, that is performance guarantees. There are certain types of problems for QAOA, certain instances of certain problems where you know what the performance is, or you know what the optimized variational parameters are from an analytical result. What this all speaks to is that when you are looking at these quantum computers and doing benchmarking, you really need to build up a library of benchmarks that are instant and problem specific and that look at the solution type in these limits that I mentioned: *heuristic, approximate, exact*. I think through those kinds of studies, we can learn better about how this algorithm behaves. But right now, it's a very heuristic kind of experimentation.

Dan: Makes sense. As a burgeoning researcher in this field or a practitioner of sorts, like who would I be that would actually pay attention to this, or what would my takeaway be from this? Are all quantum researchers going to be interested in this, or is it going to be a narrow area, and why would I be interested?

Jason: I don't think all would be interested. That is too much. But I think people who are looking at this NISQ era, which seems to be dominated by variational algorithms, like, I don't know, 90 plus percent of the suggested algorithms in this era are of a variational nature. I think the things that we found in the paper are of a general interest to people studying these algorithms. I think some of the strategies we used, the methodology, is applicable regardless of what the variational algorithm is or what the actual application/problem is that's trying to be solved. We're working now with our coworker Ben on his variational Hamiltonian diagonalization [VHD] algorithm, another variational algorithm. I have been having very similar conversations with him about that algorithm and other algorithms like VQE [variational quantum eigensolver] and trying to see if the things that we found are applicable there.

Dan: Right, and that leads us into, where are we taking this research? What's next?

Jason: I think one thing we'll probably do with this research is for QAOA, we added noise to our analysis. I think we also want to expand the types of problems. So going beyond MaxCut to other types of problems, and then I just mentioned Ben's VHD work. Then we are involved in other work with VQE, solving molecular systems. I think that is the direction we are going in because we want to expand beyond this application. This is really a combinatorial optimization problem.

Dan: Right.

Jason: Then we also, we are aware of this DARPA onus project, where there's an entire project of people examining these types of problems. So, I think that if we do continue this work with QAOA, we're going to move beyond MaxCut, or already have, and add a noise to the analysis. But then also as a group, we're moving more towards Hamiltonian simulation. I think that has a better chance of showing some advantage in this NISQ era.

Dan: Could you define NISQ era again one more time?

Jason: Oh, yes, so it's just noisy intermediate-scale quantum era, quantum computing era.

Dan: Right, right, and this is probably defined by computers that we can simulate using classical computers, right?

Jason: Not fully defined like that. That is the situation. Most of them that exist are classically simulatable. Then the NISQ era just means before we get active error correction. If you are using actual physical qubits and physical gate operations for the algorithm, I think that is what defines a NISQ era.

Dan: OK.

Jason: I don't know what the limit is on NISQ era. This all depends on the particular architectural design and how that team plans to implement error correction. Like IBM and Google, their NISQ era apparently is going to last all the way up to NISQ chips that are like 1,000 qubits in size. That is their target for when, *Oh, now we will have enough qubits, and we can run large enough distance error correction that we can produce a very high-tolerance, high-fidelity qubit*. Whereas IQM thinks—well, because they think they can produce physical qubits and get operations with higher fidelity, but they will be able to use much less of them in a quantum era correction code. So maybe their NISQ era is only going to last till 100 qubits instead of 1,000.

Dan: So all these algorithms, VQE, VHD, QAOA, they can exist before that error corrected scheme comes into play?

Jason: Yes.

Dan: Now, is this the same for, you were talking about Hamiltonian simulation too.

Jason: Yes.

Dan: What is that? Is that a NISQ era algorithm? Or is that a post NISQ era?

Jason: It's both, and there are two different ways to do it depending on what era you are in. So, Ben's VHD algorithm is an attempt to be able to do Hamiltonian simulation in the NISQ era. But then there is a separate algorithm and scheme for doing it if you are in a fault-tolerant regime. You have to use tolerized 2,100 QPE [quantum phase estimation]. The question might be like, *Can you use variational and NISQ-era algorithms on fault-tolerant devices?* I think the answer to that is yes. But the question is, *Does that provide any advantage, or would you want to, as opposed to running fault-tolerant algorithms?*

Dan: I see what you are saying. My initial instinct to that would be that if you run QAOA on a fault-tolerant quantum computer, then it would be similar to what you got in your paper, right? The result should be similar to that because there is no noise or very limited noise, and so even then you wouldn't have to simulate anymore.

Jason: This came up during a round of the review where I wanted to put in a discussion on this topic, which was like, *OK, we simulate this algorithm in a noiseless simulator, and we get these results. What can you say about how those results relate to if you were in a fault-tolerant regime?* OK, but the problem with fault-tolerant regime is the large overhead associated with doing operations. So a given operation in the fault-tolerant regime might take… We made assumptions in the paper about what the physical gate operations would be according to the assumptions we made about the architecture. You can basically take all those gate times we used and increase them by like two or three orders of magnitude because that is how much time it would cost to do the fault-tolerant operation.

Dan: Right.

Jason: That would totally ruin the performance. If you look at the performance in our comparison with classical, we were basically competing with classical, assuming fast gate operations for a NISQ device. But if you had to all the sudden use fault-tolerant operations, then that overhead is going to kick back up by two or three orders of magnitude at least. That is going to make this not performant with classical again, unless it scales. So another thing that we are able to do is really look at how this algorithm performs for very large instance sizes, hundreds of nodes, thousands of nodes, and more. To do that would have taken a lot more work and computation than we had available to us and, also, it may end up taking just actual benchmarks on real devices. You bring up a good point. It seems like if these are noiseless results, they are most appropriate in a fault-tolerant regime. It's like, *Yes, but then you got to add in the overhead to the assumptions we made on the gate operations, and that's going to take the time to solution way up*.

Dan: Perhaps if I were to resay it now, I would say that the quality of solution from your paper would probably be accurate, but the time to solution would be way worse, then you would not be competing with classical.

Jason: Yes, and there is a very good paper. We can link it. I think we are making a transcript. We can link this paper that basically, but I forget who did it, but I think they were associated with Google. But they looked at basically in the fault-tolerant regime, what is the overhead associated with compiling like generic QAOA circuits to fault-tolerant computers. Their findings were essentially what I am talking about, which is that these fault-tolerant operations are orders of magnitude more costly than the native get operations, the physical operations. It basically makes it so that you won't be performant with classical computing on these types of problems.

Dan: OK.

Jason: I think their quote in their abstract was, *If you take the Sherrington-Kirkpatrick (SK) model, what would take like four minutes on a laptop today would take a whole day on a fault-tolerant quantum computer*. This goes back to the idea that, at least with our current fault-tolerant designs, because there is so much overhead, you have to focus on problems where you expect a super polynomial speed-up over classical. You need that difference to overcome this giant overhead.

Dan: The other solution to that is to just… It is an engineering solution at the hardware level, right? If we could perhaps make the operations better, better error correction, then the algorithms won't have to work so hard. They won't have to be exponential in nature. They could be quadratic in nature.

Jason: Yes, that would be very nice. If someone could figure out a better way of implementing error correction rather than like the surface code.

Dan: Yes, because it would be nice to be able to not have to always rely on exponential incrementation, right? Something like quadradic.

Jason: Yes. You would go to quantumalgorithmzoo.org and have all 60 of them, or whatever the number is, 60-something algorithms, not just a couple.

Dan: Anything else that you would like to go over in the paper that you would like to share with anyone?

Jason: Seems like we covered a bunch.

Dan: I think we did.

Jason: Most of it. Yes. I can't think of anything else.

Dan: OK, perfect. Well then, with that, Jason, thank you for joining me today to talk about this work. To our listeners, thank you for joining us today. We will include the links in our transcript to all of our resources, including the paper mentioned on this podcast. Thank you very much, and we will see you again.

Jason: Thank you too, man.