Quantum Programming
Algorithms: Approximate Optimization

Jens Palsberg

May 13, 2019

Outline

Hook: Will quantum computing win? At what? Our next quantum algorithm is the quantum approximate optimization algorithm, also known as QAOA. It solves a natural problem and is currently the best hope for a quantum advantage of real quantum computers in the next five years.

Purpose: Persuade you that you can understand this algorithm.

Preview:

1. We will make sure that everybody understands the problem.
2. We will show how QAOA searches a grid of options to find a good one.
3. We will cover full details of why QAOA works.

Transition to Body: Now let us talk about the problem that this algorithm solves.

Main Point 1: We will make sure that everybody understands the problem.
[The problem is MaxSAT and the goal of QAOA is to solve it approximately]
[The problem is NP-complete]
[QAOA has inspired researchers to design better classical algorithms]

Transition to MP2: Now let us look at how to solve the problem on a quantum computer.

Main Point 2: We will show how QAOA searches a grid of options to find a good one.
[The algorithm represents the constraints as a function that marks the good cases]
[The algorithm uses two parameters to create a search space]
[The idea is to vary the two parameters and pick the best result]

Transition to MP3: Now let us get into what each iteration is doing.

Main Point 3: We will cover full details of why QAOA works.
[The idea of the first rotation matrix is to separate the good cases]
[The idea of the second rotation matrix is to boost the amplitude of the good cases]
[Those ideas can be enhanced by using more parameters and more matrices]

Transition to Close: So there you have it.

Review: QAOA searches a grid of options to find a good approximation of the optimal solution with high probability.

Strong finish: This year, DARPA is putting $33 million into finding out whether QAOA on quantum computers is faster than classical algorithms on classical computers. If this effort succeeds, it will be a watershed moment for quantum computing.

Call to action: Think about other NP-complete problems that variations of QAOA can solve.

Detailed presentation

Hook: Will quantum computing win? At what? Our next quantum algorithm is the quantum approximate optimization algorithm, also known as QAOA. It solves a natural problem and is currently the best hope for a quantum advantage of real quantum computers in the next five years.

QAOA solves an NP-complete problem approximately. This is interesting because, at scale, NP-complete problems are intractable for classical computers. Many NP-complete problems are important in practice so classical computers solve them approximately. The question is: can QAOA give better approximate solutions than classical computers within the same time frame, or give as good solutions faster?

Purpose: Persuade you that you can understand this algorithm.

Preview:

1. We will make sure that everybody understands the problem.
2. We will show how QAOA searches a grid of options to find a good one.
3. We will cover full details of why QAOA works.

Transition to Body: Now let us talk about the problem that this algorithm solves.

Main Point 1: We will make sure that everybody understands the problem.

[The problem is MaxSAT and the goal of QAOA is to solve it approximately]
The MaxSAT problem is: given a Boolean formula that is a conjunction of some clauses, satisfy as many of the clauses as possible. We will focus on restricted form of MaxSAT that has all the essential computational challenges of the general problem. This restricted form is called Max2SAT and has the following definition.

The Max2SAT problem:
Intuition: n variables, m clauses, satisfy as many clauses as possible.
Input: an integer $t \in 1..m$ and a Boolean formula $\bigwedge_{j=1}^{m} (\ell_{j1} \vee \ell_{j2})$.
Terminology: we call each $(\ell_{j1} \vee \ell_{j2})$ a clause, and we call each ℓ_{jq} a literal.
Notation: each literal ℓ_{jq} is either a variable x_p or the negation of a variable, written $\neg x_p$.
Convention: the variables form the set $x_0, \dots, x_{n-1}$.
Notation: we use z to range over mappings from variables to Booleans.
Output: 1 if $\exists z$: z satisfies at least t of the clauses, and 0 otherwise.

Here is an example instance of the Max2SAT problem.

$$\begin{aligned} t = 8 \qquad & (x_1 \vee x_2) \wedge (x_2 \vee \neg x_5) \wedge (\neg x_3 \vee \neg x_4) \wedge \\ & (x_2 \vee \neg x_6) \wedge (\neg x_3 \vee \neg x_5) \wedge (x_3 \vee \neg x_4) \wedge \\ & (\neg x_4 \vee x_5) \wedge (x_3 \vee \neg x_0) \wedge (x_4 \vee x_0) \end{aligned}$$

Here we have a formula with 9 clauses and the goal to satisfy at least 8 of them. Notice that the formula uses 7 variables, called $x_0, \dots, x_6$.

[The problem is NP-complete]
Intuitively, Max2SAT is NP-complete because we have no easy way to satisfy more and more clauses

in an efficient way. Specifically, if we try a strategy that tries to satisfy more and more clauses, we can get stuck in a local optimum and miss the best solution.

In an attempt to solve the problem instance above, we might begin with a random assignment. For example, suppose we begin with the assignment that assigns true to every variable. Now we can evaluate how many clauses this assignment satisfies, and we will find that it satisfies 7 clauses. How do improve this assignment to a different one that satisfies 8 clauses? Perhaps we can think of something by studying this particular formula, like we could change the assignment of x_4 from true to false. Indeed, this change would give us an assignment that satisfies 8 clauses. However, in general, we have no efficient strategy for doing such an improvement in a way that gets us to a global maximum or beats a given t.

[QAOA has inspired researchers to design better classical algorithms]
QAOA solves MaxSAT approximately and after in came out in 2014, it was for a while the best algorithm on any kind of computer. Then in 2015 the classical-algorithms people upped their game and presented a new classical algorithm that beats QAOA in certain ways. However, in 2016 the QAOA people published a paper that argues that QAOA may give a quantum advantage over classical approximation algorithms. In 2019, a group of researchers published an improvement of QAOA.

Transition to MP2: Now let us look at how to solve the problem on a quantum computer.

Main Point 2: We will show how QAOA searches a grid of options to find a good one.

[The algorithm represents the constraints as a function that marks the good cases]
Let us represent a mapping z as a bit string in $\{0,1\}^n$ with the idea that z maps x_k to the k'th bit in the bit string. Henceforth, when we write z, we mean the bit string representation.

Given $z \in \{0,1\}^n$ and a Boolean formula $\bigwedge_{j=1}^{m} (\ell_{j1} \vee \ell_{j2})$, define

$$
\begin{aligned}
\mathit{Count}_j(z) &= \begin{cases} 1 & \text{if } z \text{ satisfies the } j\text{'th clause} \\ 0 & \text{otherwise} \end{cases} \\[1em]
\mathit{Count}(z) &= \Sigma_{j=1}^{m}\ \mathit{Count}_j(z) \\[1em]
C_j|z\rangle &= \mathit{Count}_j(z)\ |z\rangle \\
C|z\rangle &= \mathit{Count}(z)\ |z\rangle \ = \ \Sigma_{j=1}^{m}\ \mathit{Count}_j(z)\ |z\rangle \ = \ \Sigma_{j=1}^{m}\ C_j|z\rangle \\[1em]
B &= \Sigma_{k=0}^{n-1}\ NOT_k \\
&= NOT \otimes I^{\otimes(n-1)} \ + \ I \otimes NOT \otimes I^{\otimes(n-2)} \ + \ \ldots \ + \ I^{\otimes(n-1)} \otimes NOT
\end{aligned}
$$

Notice that *Count* is a function in the classical world: it maps bit strings to numbers. In contrast, C is a function in the quantum world: it maps quantum states to quantum states (well, unnormalized quantum states). The idea is that C uses *Count* to mark the good cases. Specifically, in $C|z\rangle$, the amplitude of $|z\rangle$ is the number of clauses that z satisfies.

In the definition of B, we use NOT_k as informal notation to mean application of NOT to the k'th qubit.

From C we will build a separator and from B we will build a mixer.

[The algorithm uses two parameters to create a search space]
The algorithm uses two parameters $(\gamma, \beta) \in [0, 2\pi] \times [0, \pi]$ to get variation in the results of the runs. From C, B, γ, β, we build two rotation matrices: $\mathit{Sep}(\gamma) = e^{-i\gamma C}$ and $\mathit{Mix}(\beta) = e^{-i\beta B}$.

[The idea is to vary the two parameters and pick the best result]
Here is QAOA:

```
for many different choices of $(\gamma, \beta) \in [0, 2\pi] \times [0, \pi]$ {
    measure $\mathit{Mix}(\beta)\ \mathit{Sep}(\gamma)\ H^{\otimes n}\ |0^n\rangle$
}
pick the measurement $z$ that maximizes $\mathit{Count}(z)$.
```

Here is the intuition behind how the algorithm works.

First we use $H^{\otimes n}$ to create a uniform superposition of all bit strings. The idea of a uniform superposition is that every bit string has the same chance of being measured.

Then we use $\mathit{Sep}(\gamma)$ to mark the good cases with special amplitudes, while maintaining the uniform superposition.

Finally, we use $\mathit{Mix}(\beta)$ to boost the amplitudes of the good cases.

Transition to MP3: Now let us get into what each iteration is doing.

Main Point 3: We will cover full details of why QAOA works.

[The idea of the first rotation matrix is to separate the good cases]
The separator matrix is $\mathit{Sep}(\gamma) = e^{-i\gamma C}$. Notice that C is a diagonal matrix, hence C Hermetian, hence $e^{-i\gamma C}$ is a unitary matrix. Indeed, $e^{-i\gamma C}$ is a rotation matrix. Let us calculate to see some details of $\mathit{Sep}(\gamma)$.

$$\begin{aligned} \mathit{Sep}(\gamma) &= e^{-i\gamma C} \\ &= e^{-i\gamma \Sigma_{j=1}^m\ C_j} \\ &= \Pi_{j=1}^m\ e^{-i\gamma C_j} \end{aligned}$$

The third step is called Trotter decomposition and is correct because the C_j matrices commute.

Let us do a case analysis of $e^{-i\gamma C_j}$. The analysis begins with considering C_j. Recall:

$$\mathit{Count}_j(z) = \begin{cases} 1 & \text{if } z \text{ satisfies the } j\text{'th clause} \\ 0 & \text{otherwise} \end{cases}$$

$$C_j|z\rangle = \mathit{Count}_j(z)\ |z\rangle$$

We have two cases, 0 and 1, of Count_j, so we have two cases of C_j:

$$\begin{aligned} C_j|z\rangle &= \begin{cases} |z\rangle & \text{if } \mathit{Count}_j(z) = 1 \\ \mathbf{0} & \text{if } \mathit{Count}_j(z) = 0 \end{cases} \\ e^{-i\gamma C_j}|z\rangle &= \begin{cases} e^{-i\gamma}\ I & \text{if } \mathit{Count}_j(z) = 1 \\ I & \text{if } \mathit{Count}_j(z) = 0 \end{cases} \end{aligned}$$

Above we use I to denote the identity matrix and we use $\mathbf{0}$ to denote the vector that is all 0.

Now we can continue the calculation of $Sep(\gamma)$:

$$\begin{aligned} Sep(\gamma)\ \Sigma_{z\in\{0,1\}^n}\ a_j|z\rangle &= \Pi_{j=1}^m\ e^{-i\gamma C_j}\ \Sigma_{z\in\{0,1\}^n}\ a_z|z\rangle \\ &= \Sigma_{z\in\{0,1\}^n}\ a_z\ \Pi_{j=1}^m\ e^{-i\gamma C_j}\ |z\rangle \\ &= \Sigma_{z\in\{0,1\}^n}\ a_z\ (e^{-i\gamma})^{Count(z)}\ |z\rangle \end{aligned}$$

So, $|z\rangle$ gets a factor $e^{-i\gamma}$ for each clause that z satisfies.

The next task is to design a circuit for $Sep(\gamma)$. We will do that by adding a helper qubit.

We will give an example of the construction of the circuit. Consider the following formula with two variables and two clauses:

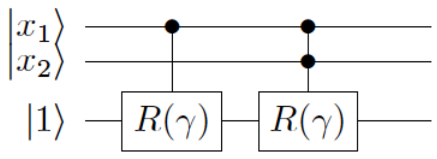
$$x_0\ \wedge\ (x_0 \wedge x_1)$$

We would have some work to do to get this formula into the format that we used at the beginning of the lecture, but let us ignore that. We have:

$$\begin{aligned} & Sep(\gamma)\ (a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle) \\ = \quad & a_{00}|00\rangle + a_{01}|01\rangle + (e^{-i\gamma})^1 a_{10}|10\rangle + (e^{-i\gamma})^2 a_{11}|11\rangle \end{aligned}$$

Notice that we changed the amplitude of the good cases.

We can get this effect via the following circuit:



Here,

$$R(\gamma) \quad = \quad \begin{pmatrix} 1 & 0 \\ 0 & e^{-i\gamma} \end{pmatrix}$$

Now we can do the calculation again with this circuit as $Sep(\gamma)$ and with a helper qubit.

$$\begin{aligned} & Sep(\gamma)\ (a_{00}|00\rangle \otimes |1\rangle + a_{01}|01\rangle \otimes |1\rangle + a_{10}|10\rangle \otimes |1\rangle + a_{11}|11\rangle) \otimes |1\rangle \\ = \quad & a_{00}|00\rangle \otimes |1\rangle + a_{01}|01\rangle \otimes |1\rangle + a_{10}|10\rangle \otimes (e^{-i\gamma})^1|1\rangle + a_{11}|11\rangle \otimes (e^{-i\gamma})^2|1\rangle \\ = \quad & a_{00}|001\rangle + a_{01}|011\rangle + (e^{-i\gamma})^1 a_{10}|101\rangle + (e^{-i\gamma})^2 a_{11}|111\rangle \end{aligned}$$

For example, let us consider $\gamma = \frac{\pi}{4}$. We have

$$\begin{aligned} e^{-i\gamma} &= e^{i(-\gamma)} = \cos(-\gamma) + i\sin(-\gamma) = \cos(\gamma) - i\sin(\gamma) = \frac{1}{\sqrt{2}} - i\frac{1}{\sqrt{2}} \\ (e^{-i\gamma})^2 &= (\frac{1}{\sqrt{2}} - i\frac{1}{\sqrt{2}})^2 = \frac{1}{2}(1-i)^2 = \frac{1}{2}(-2i) = -i \end{aligned}$$

[The idea of the second rotation matrix is to boost the amplitude of the good cases]
The mixer matrix is $\mathit{Mix}(\beta) = e^{-i\beta B}$. Recall:

$$B = \Sigma_{k=0}^{n-1} \; NOT_k$$

We can look up a formula for exponentiation of matrices and find that

$$\begin{aligned} \mathit{Mix}(\beta) &= e^{-i\beta B} \\ &= e^{-i\beta \; \Sigma_{k=0}^{n-1} \; NOT_k} \\ &= \Pi_{k=0}^{n-1} \; e^{-i\beta NOT_k} \\ &= (\otimes n) \; R_x(2\beta) \end{aligned}$$

Here,

$$R_x(2\beta) = \begin{pmatrix} \cos(\beta) & -i\sin(\beta) \\ -i\sin(\beta) & \cos(\beta) \end{pmatrix}$$

For example, for $\beta = \frac{\pi}{2}$:

$$R_x(2\frac{\pi}{2}) = \begin{pmatrix} 0 & -i \\ -i & 0 \end{pmatrix} = -i \; NOT$$

Another example, for $\beta = \frac{\pi}{4}$:

$$R_x(2\frac{\pi}{4}) = \begin{pmatrix} \frac{1}{\sqrt{2}} & -i\frac{1}{\sqrt{2}} \\ -i\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} = \frac{1}{\sqrt{2}} \; (I - i \; NOT)$$

Another example, for $\beta = \frac{\pi}{8}$:

$$R_x(2\frac{\pi}{8}) = \begin{pmatrix} \frac{1}{2}\sqrt{2+\sqrt{2}} & -i\frac{1}{2}\sqrt{2-\sqrt{2}} \\ -i\frac{1}{2}\sqrt{2-\sqrt{2}} & \frac{1}{2}\sqrt{2+\sqrt{2}} \end{pmatrix} = \frac{1}{2}\begin{pmatrix} \sqrt{2+\sqrt{2}} & -i\sqrt{2-\sqrt{2}} \\ -i\sqrt{2-\sqrt{2}} & \sqrt{2+\sqrt{2}} \end{pmatrix}$$

Now let us do a run of QAOA on the Boolean formula from earlier:

$$x_0 \; \wedge \; (x_0 \wedge x_1)$$

We will do a single iteration of the for-loop with a single choice of (γ, β). We will pick $\gamma = \frac{\pi}{4}$ and $\beta = \frac{\pi}{4}$.

We will ignore the helper qubit that we need along the way.

$$
\begin{aligned}
& Mix(\beta)\ Sep(\gamma)\ H^{\otimes 2}\ |00\rangle \\
= \ & Mix(\beta)\ Sep(\gamma)\ \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) \\
= \ & Mix(\beta)\ \frac{1}{2}(|00\rangle + |01\rangle + (\frac{1}{\sqrt{2}} - i\frac{1}{\sqrt{2}})|10\rangle - i|11\rangle) \\
= \ & (\frac{1}{\sqrt{2}}\ (I - i\ NOT) \otimes \frac{1}{\sqrt{2}}\ (I - i\ NOT))\frac{1}{2}(|00\rangle + |01\rangle + (\frac{1}{\sqrt{2}} - i\frac{1}{\sqrt{2}})|10\rangle - i|11\rangle) \\
= \ & \frac{1}{4}((I - i\ NOT) \otimes (I - i\ NOT))(|00\rangle + |01\rangle + (\frac{1}{\sqrt{2}} - i\frac{1}{\sqrt{2}})|10\rangle - i|11\rangle) \\
= \ & \frac{1}{4}((|00\rangle + |01\rangle + (\frac{1}{\sqrt{2}} - i\frac{1}{\sqrt{2}})|10\rangle - i|11\rangle)\ +\ (-i|10\rangle - i|11\rangle + (-i\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}})|00\rangle - |01\rangle)\ + \\
& \quad (-i|01\rangle - i|00\rangle + (-i\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}})|11\rangle - |10\rangle)\ +\ (-|11\rangle - |10\rangle - (\frac{1}{\sqrt{2}} - i\frac{1}{\sqrt{2}})|01\rangle + i|00\rangle)) \\
= \ & \frac{1}{4}((1 + (-i\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}))|00\rangle\ +\ (-i - (\frac{1}{\sqrt{2}} - i\frac{1}{\sqrt{2}}))|01\rangle\ + \\
& \quad ((\frac{1}{\sqrt{2}} - i\frac{1}{\sqrt{2}}) - i - 2)|10\rangle\ +\ (-2i + (-i\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}) - 1)|11\rangle) \\
= \ & \frac{1}{4}(((1 - \frac{1}{\sqrt{2}}) - \frac{1}{\sqrt{2}}i)|00\rangle\ +\ (-\frac{1}{\sqrt{2}} - (1 - \frac{1}{\sqrt{2}})i)|01\rangle\ + \\
& \quad ((-2 + \frac{1}{\sqrt{2}}) - (1 + \frac{1}{\sqrt{2}})i)|10\rangle\ +\ (-(1 + \frac{1}{\sqrt{2}}) - (2 + \frac{1}{\sqrt{2}})i)|11\rangle)
\end{aligned}
$$

Now we can calculate the probabilities of measuring each of the four cases:

$$
00\ :\ \frac{2 - \frac{2}{\sqrt{2}}}{16} \qquad 01\ :\ \frac{2 - \frac{2}{\sqrt{2}}}{16} \qquad 10\ :\ \frac{6 - \frac{2}{\sqrt{2}}}{16} \qquad 11\ :\ \frac{6 + \frac{6}{\sqrt{2}}}{16}
$$

Here we can see that the two bit strings (00, 01) that satisfy 0 clauses each has low probability, while the bit string (10) that satisfies 1 clause has much higher probability, and the bit string (11) that satisfies 2 clauses has the highest probability. So, chances are that we will run QAOA on the formula, measure 11, check that 11 satisfies both clauses, and be done.

[Those ideas can be enhanced by using more parameters and more matrices]
QAOA has a more general version that in each iteration does p applications of $(Mix(\beta)\ Sep(\gamma))$. Each application uses a different pair (γ, β). In the limit $(p \to \infty)$, this gives the best solution.

Transition to Close: So there you have it.

Review: QAOA searches a grid of options to find a good approximation of the optimal solution with high probability.

Strong finish: This year, DARPA is putting $33 million into finding out whether QAOA on quantum computers is faster than classical algorithms on classical computers. If this effort succeeds, it will be a watershed moment for quantum computing.

Call to action: Think about other NP-complete problems that variations of QAOA can solve.