QAOA PROJECT

TOM WAKNINE

1. Generic QAOA algorithm

1.1 Introduction

In The field of combinatorial optimization we often try to optimize some objective function, or at the very least approximate an optimize solution. Formally we make the following definitions

Definition 1.1. C_α will be a clause of n bits (or just a clause) if it is a function $C_\alpha : \{0,1\}^n \to \{0,1\}$. We will call α a condition and say that $z \in \{0,1\}^n$ satisfy α if $C_\alpha(z) = 1$

Definition 1.2. C is objective function with m clause and n bits if it is of the form

$$C(z) = \sum_{\alpha=1}^{m} C_\alpha(z)$$

where $z \in \{0,1\}^n$ and each of the C_α is a clause of n bits

With those definition the satisfiability problem ask us to determine if there is some $z \in \{0,1\}^n$ that satisfy all the conditions, i.e. if there is such z such that $C(z) = m$. The MaxSat problem ask us to find a string $z \in \{0,1\}^n$ such that $C(z) = \max_{z' \in \{0,1\}^n} C(z')$. Finally the Approximate optimization problem ask us to give z such that $C(z)$ is close to the maximum, this not very formal but in different versions of the problem we may make different requirements on the error term $|C(z) - \max_{z' \in \{0,1\}^n} C(z')|$. Now for the Quantum approach toward combinatorial optimization we need a way to think on objective function and clauses (or objective functions) as operators, which we do in the following way

Definition 1.3. for any clause C_α of n bits it's Hamiltonian H_α will be

$$H_\alpha = \sum_{z \in \{0,1\}^n} C_\alpha(x) \left|x\right\rangle \left\langle x\right|$$

which is a diagonal operator over the 2^n dimensional Hilbert space by

Now defintion 1.3 is useful as it allow us to associate a function over binary string with a diagonal operator, thus it connect combinatorial objects with quantum ones. We will extend it as follow

Definition 1.4. The problem Hamiltonian H_P is the extension of 1.3 to the objective function C

$$H_P = \sum_\alpha H_\alpha = \sum_{x \in \{0,1\}^n} C(x) \left|x\right\rangle \left\langle x\right|$$

Now motivated by our will to work with unitary operators, as well as our familiarity with other problems involving Hamiltonians we may wish to exponentiate our problem Hamiltonian. And indeed it will be useful to look on the following

Definition 1.5.

$$U(H_p, \gamma) = e^{-i\gamma H_P}$$

Now it turns out we will also need the mixing Hamiltonian and it's associate unitary operator

Definition 1.6. $H_B = \sum_{i=1}^{n} \sigma_i^x$ is the mixing Hamiltonian and its unitary is

$$U(H_B, \beta) = e^{-i\beta H_B}$$

Now note that all the egienvalues of H_P are natural number, hence we may restrict γ to $[0, 2\pi]$. and similarly we restrict β to $[0, \pi]$. We will use our unitaries $U(H_P, \gamma)$, $U(H_B, \beta)$ to define a classical optimization problem as follow

Definition 1.7. given an initial state s and a sequence of angels $\boldsymbol{\gamma} \in [0, 2\pi]^p$, $\boldsymbol{\beta} \in [0, \pi]^p$ its angle dependent quantum state is

$$|\psi(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle = \prod_{i=0}^{p-1} \Big(U(H_P, \gamma_i) U(H_B, \beta_i) \Big) |s\rangle = U(H_P, \gamma_p) U(H_B, \beta_p) \dots U(H_P, \gamma_1) U(H_B, \beta_1) |s\rangle$$

Where we usually take the inital state s to be the fully mixing one

$$s = \frac{1}{\sqrt{2^n}} \sum_{z \in \{0,1\}^n} |z\rangle$$

And with that we may finally define

Definition 1.8. $F_p(\boldsymbol{\gamma}, \boldsymbol{\beta}) = \langle \psi(\boldsymbol{\gamma}, \boldsymbol{\beta})| H_P |\psi(\boldsymbol{\gamma}, \boldsymbol{\beta})\rangle$ is the expectation of H_P with our state and $M_p = \max_{\boldsymbol{\gamma}, \boldsymbol{\beta}} F_p(\boldsymbol{\gamma}, \boldsymbol{\beta})$ is its optimization.

Note that an equivalent way to evaluate F at $\boldsymbol{\gamma}, \boldsymbol{\beta}$ is to measure $\psi(\boldsymbol{\gamma}, \boldsymbol{\beta})$ in the standard basis to get a string z and then $C(z)$ will be the value of F with those angels. It was shown in [1] that we have

$$\lim_{p \to \infty} M_p = \max_{z \in \{0,1\}^n} C(z)$$

Now we want to construct our Hamiltonian in the Pauli Z operators. to do that we use the assumption that each clause depends only on a Small number of bits, i.e for each α there are some $Q_\alpha, \overline{Q}_\alpha \subset [n]$ such that $C_\alpha(z) \neq 0$ iff $z_i = 1$ for all $i \in Q_\alpha$, $z_i = 0$ for all $i \in \overline{Q}_\alpha$ and $|Q_\alpha| + |\overline{Q}_\alpha|$ is relatively small for all α. Then we can write our objective function in the canonical form

$$C(z) = \sum_{\alpha=1}^{m} w_\alpha \prod_{i \in Q_\alpha} z_i \prod_{i \in \overline{Q}_\alpha} (1 - z_i)$$

where we use w_α to generalize our construction to clauses C_α that may get values other then 1. And so using the map $x_i \to \frac{1}{2}(1 - Z_i)$ we can write our Hamiltonian as

$$H_P = \sum_{\alpha=1}^{m} \frac{w_\alpha}{2^{|Q_\alpha|+|\overline{Q}_\alpha|}} \prod_{i \in Q_\alpha} (1 - Z_i) \prod_{i \in \overline{Q}_\alpha} (1 + Z_i)$$

Hence using quantum information we were able to transform our combinatorial optimization problem to a classical optimization problem. In particular we note that after optimizing F_{p-1} to be close to M_{p-1} we only need to optimize the two new angels γ_p, β_p in F_p to approximate M_p. And so if we let *Opt* be some iterative classical optimization algorithm we can get the following algorithm to solve our Approximate optimization problem

Where the condition is step 2 is depending on the classical optimization algorithm we choose

Algorithm 1 generic QAOA algorithm

1: Generate $U(H_P, \gamma)$, $U(H_B, \beta)$ from our combinatorial problem and initialize $\boldsymbol{\gamma}, \boldsymbol{\beta}$ with some starting value
2: **while** *Opt* not done **do**
3: measure $\psi(\boldsymbol{\gamma}, \boldsymbol{\beta})$ in the standard basis to generate sting z and evaluate $C(z)$.
4: $(\boldsymbol{\gamma}, \boldsymbol{\beta}) \leftarrow Opt(C(z))$
5: **end while**
6: Measure $\psi(\boldsymbol{\gamma}, \boldsymbol{\beta})$ to get a string z for which $C(z) \sim M_p \sim \max_{z' \in 0,1^n} C(z')$
7: **return** z

as well as our desired accuracy. Now there are some thing we need to specify for the above algorithm. Specifically which algorithm we want to use as *Opt* and how do we chose the starting value of the angles, as well as how do we choose p to be big enough such that M_p will be close to the maximal value. But non the less the above give a good overview of our generic QAOA algorithm. Note that as p goes to ∞ the probability that our measurement will produce a string z that optimize our problem increases to 1. Hence assuming that our problem is NP, i.e. given a string we can determine if it optimize our objective function we get that the above algorithm under some conditions will actually solve the MaxSat problem and not just the Approximate optimization one(just run it with big p again and again and each time check if the resulting string is maximize C).

1.2 Classical optimization algorithm

There are many classical algorithm that find the maximum of a function f by iteratively evaluating it at a point and readjusting. The most famous of which is the Newton's Raphson method, which look on the iterative process $x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$ that on certain condition will converge to some extremum point of f. Now this method has many generalization, for example in multivariate case where we can use the iterations $x_{k+1} = x_k - H^{-1}(x_k)\Delta f(x_k)$, where H is the Hessian matrix. In those types of algorithms a recurring theme of those generalization is the use of the derivative (or it's multivariate counterpart). Note that in the above if we wanted to find an extremum point of f we need to know both f' and f'' (or Δf and H in the multivariate case) so we need to know

the second derivative of f. Those class of methods are called second derivative methods or Newton's method, and in general they tend to converge faster then first derivative methods. This is because knowing the derivative tell us only in which direction we should move, but the second derivative tell us what the step size should be.

The problem with those kind of methods is that they require us to have some non trivial knowledge on our function (specifically it's derivatives) which often can not be computed easily. The solution usually taken in cases like that is to use the evaluations of the function in order to approximate those derivative, and then use those approximation to find the maxima of the function. algorithms that use this method of approximating the derivative are called quasi-Newton methods or semi-Newton methods.

1.3 **BFGS** the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm is one such quasi-Newton method which use iteration to both approximate the Hessian matrix H of f and it's minima (or maxima). The BFGS algorithm start from some initional guess B_0, x_0 of the Hessian matrix and the minima and from there doing the following iterations and doing may

Algorithm 2 BFGS algorithm

1: find $p_k \in \mathbb{R}^n$ by solving $B_k p_k = -\Delta f(x_k)$
2: use one dimensional optimization to find $\alpha_k \in \mathbb{R}$ that (approximately) minimize the function $\alpha \to f(x_k + \alpha p_k)$
3: set $s_k = \alpha_k p_k$ and update $x_{k+1} = x_k + s_k$
4: set $y_k = \Delta f(x_{k+1}) - f(x_k)$
5: update $B_{k+1} = B_k + \frac{|y_k\rangle\langle y_k|}{\langle y_k|s_k\rangle} - \frac{|B_k s_k\rangle\langle B_k s_k|}{\langle s_k|B_k|s_k\rangle}$

iteration of the above will give us B_k that is close to H and more impotently x_k that is close to the minima of f.

2. The VQF algorithm

2.1 **Introduction** We will now turn to use of QAOA algorithm to solve the problem of integer factorization, i.e given some number m finding its prime factor. specifically we will assume that $m = p \cdot q$ for some primes p, q (since it is easy to reduce to that case). For that we will consider the binary representation of our numbers

$$m = \sum_{i=0}^{n_m} m_k 2^i$$
$$p = \sum_{i=0}^{n_p} p_i 2^i$$
$$q = \sum_{i=0}^{n_q} q_i 2^i$$

So we can turn the equation $m = p \cdot q$ into

$$\sum_{k=0}^{n_m} m_k 2^k = \Big(p = \sum_{i=0}^{n_p} p_i 2^i\Big)\Big(q = \sum_{i=0}^{n_q} q_i 2^i\Big) = \sum_{k=0}^{n_p+n_q} \sum_{i=0}^{k} p_i q_{k-i} 2^k \tag{2.1}$$

Now we want to turn this into a set of equation (which will be our clauses), but note that due to the carry bit the above dose **not** imply that $m_k = \sum_{i=0}^{k} p_i q_{k-i}$. So let us introduce those carry bit into our equation. we define the carry bits into the i coefficient by induction, with $w_0 = 0$ (since there is no carry bit from the first muktiplication), and then assuming we define w_j for all $j < k$ we define w_k by

$$w_k = \sum_{i=0}^{k} p_i q_{k-i} + w_{i,k-i}$$

where $w_{k,j}$ is the binary representation of w_k so

$$w_k = \sum_{j=0}^{n_k} w_{k,j} 2^j$$

And now (2.1) becomes

$$\sum_{k=0}^{n_m} m_k 2^k = \sum_{k=0}^{n_p+n_q} \sum_{i=0}^{k} \Big(p_i q_{k-i} + w_{i,k-i} - \sum_{j=1}^{n_k} w_{k,j} 2^j\Big) 2^k \tag{2.2}$$

And note that the coefficient of 2^k in the LHS is simply $w_{k,0}$ which is just a single bit (thus must be equal to m_k). Hence we can turn (2.2) into $n+p+n_q$ equations, for that we denote $n_c = n_p + n_q$ and we expend w_k into a n_c bits by setting $w_{k,j} = 0$ for $n_k < j \leq n_c$. We also make the change of variable $z_{i,j} = w_{i,j-i}$, so $z_{i,j}$ represent the carry bit from bit position i into bit position j. Then we get the following set of n_c equations

$$m_i = \sum_{j=0}^{i} p_i q_{j-i} + z_{i,j} - \sum_{k=1}^{n_c} z_{i,i+k} 2^k \tag{2.3}$$

Which allow us to define our clauses

$$C_i = \sum_{j=0}^{i} \big(p_i q_{j-i} + z_{i,j}\big) - m_i - \sum_{k=1}^{n_c} z_{i,i+k} 2^k \tag{2.4}$$

Now note that it is not clear over how may bits our clauses are defined since we don't know n_p, n_q A priori. But if we assume WOLOG that $q \leq p$ we know that $\sqrt{m} \leq q \leq p \leq m$ hence $\lfloor \frac{m}{2} \rfloor \leq n_q \leq n_p \leq n_m$ so by extending with 0 we may assume that $n_p = n_m$ and $n_q = \lfloor \frac{m}{2} \rfloor$. Now before we construct our Hamiltonian we need to make some preprocessing to simplify

the equations (2.2). For that we using the following rules

$$\begin{aligned} xy - 1 = 0 &\implies x = y = 1 \\ x + y - 1 = 0 &\implies xy = 0 \\ a - bx = 0 &\implies x = 1 \\ \sum_i x_i = 0 &\implies x_i = 0 \\ \sum_{i=1}^{a} x_i - a = 0 &\implies x_i = 1 \end{aligned}$$

And now we can let C_i' be the simplified form of C_i after using the above rules on our equations. We note that by numerical results from [2] going from C_i to C_i' will reduce the number of qubits from $O(n_m \log n_m)$ to $O(n_m)$. Now we construct our objective function, for technical reasons it will be convinent to work with $C_i'^2$ rather then C_i so we define

$$C = \sum_{i=1}^{n_m} C_i'^2 \tag{2.5}$$

And from here we may use the generic QAOA algorithm to find the solution.

References

1. Farhi, E., Goldstone, J. and Gutmann, S., 2014. A quantum approximate optimization algorithm. arXiv preprint arXiv:1411.4028.
2. Anschuetz, E., Olson, J., Aspuru-Guzik, A. and Cao, Y., 2019, March. Variational quantum factoring. In International Workshop on Quantum Technology and Optimization Problems (pp. 74-85). Springer, Cham 2
6