How Quantum Computing Can Help With (Continuous) Optimization

Christian Ayub, Martine Ceberio, and Vladik Kreinovich

Abstract It is known that the use of quantum computing can reduced the time needed for a search in an unsorted array: from the original non-quantum time T to a much smaller quantum computation time $T_q \sim \sqrt{T}$. In this paper, we show that for a continuous optimization problem, with quantum computing, we can reach almost the same speed-up: namely, we can reduce the non-quantum time T to a much shorter quantum computation time $\sqrt{T} \cdot \ln(T)$.

1 Formulation of the Problem

Known advantages of quantum computing. It is known that quantum computing enables us to drastically speed up many computations; see, e.g., [3].

One example of such a problem is the problem of looking for a given element in an unsorted n-element array. With non-quantum computations, to be sure that we have found this element, we need to spend at least n computational steps. Indeed, if we use fewer than n steps, this would mean that we only look at less than n elements of the array – and thus, we may miss the element that we are looking for.

Grover's quantum-computing algorithm [1, 2] allows us to reduce the time needed to search for an element in an unsorted array of size n from the non-quantum lower bound n to a much faster time $c \cdot \sqrt{n}$.

In other words, we reduce the time needed for this task from the non-quantum time T to a much smaller quantum computation time T_q which is proportional to $\sqrt{T}$.

Need to consider optimization problems. While search problems are ubiquitous, in many applications, we also need to solve continuous optimization problems. In

Christian Ayub, Martine Ceberio, and Vladik Kreinovich
Department of Computer Science, University of Texas at El Paso, El Paso, Texas 79968, USA
e-mail: cayub@miners.utep.edu, mceberio@utep.edu, vladik@utep.edu

such problems, we want to find an object or a strategy for which the given objective function attains its largest possible (or smallest possible) value.

An object is usually characterized by its parameters $x_1,\ldots,x_n$, for each of which we usually know the bounds $\underline{x}_i$ and $\overline{x}_i$, so that $\underline{x}_i \le x_i \le \overline{x}_i$. Let $f(x_1,\ldots,x_n)$ denote the value of the objective function corresponding to the parameters $x_1,\ldots,x_n$.

Comment. In most practical situations, the objective function is several (at least two) times continuously differentiable (smooth).

General optimization problem: idealized case. The idealized optimization problem means finding the values $x_1^{\rm opt},\ldots,x_n^{\rm opt}$ for which the function $f(x_1,\ldots,x_n)$ attains its largest possible value on the box

$$B \stackrel{\rm def}{=} [\underline{x}_1,\overline{x}_1]\times\ldots\times[\underline{x}_n,\overline{x}_n], \tag{1}$$

i.e., for which

$$f(x_1^{\rm opt},\ldots,x_n^{\rm opt}) = \max_{(x_1,\ldots,x_n)\in B} f(x_1,\ldots,x_n). \tag{2}$$

General optimization problem: practical formulation. Of course, in practice, we can only attain values approximately. So, in practice, we are looking not for the absolute maximum, but rather for the values $x_1,\ldots,x_n$ which are maximal with given accuracy $\varepsilon > 0$.

In other words, we are interested in the following problem:

- we are given a function $f(x_1,\ldots,x_n)$ defined on a given box B, and we are given a real number ε;
- we want to find values $x_1^d,\ldots,x_n^d$ for which

$$f(x_1^d,\ldots,x_n^d) \ge \left(\max_{(x_1,\ldots,x_n)\in B} f(x_1,\ldots,x_n)\right) - \varepsilon.$$

What we do in this paper. In this paper, we show that the use of quantum computing can speed up the solution of this problem as well – and we show how exactly this problem can be sped up, from non-quantum computation time T to a much shorter quantum computation time $T_q \sim \sqrt{T}\cdot\ln(T)$.

2 How This Optimization Problem Is Solved Now

We consider only guaranteed global optimization algorithms. Of course, there are many semi-heuristic ways to solve the optimization problem. For example, we can start at some point $x = (x_1,\ldots,x_n)$ and use gradient techniques to reach a *local* maximum. However, these methods only lead to a local maximum. If we want to make sure that we reached the actual (*global*) maximum, we cannot skip some subdomains of the box B, we have to analyze all of them. How can we do it?

Non-quantum lower bound. Similarly to the search problem, we can find a natural lower bound on the time complexity of a non-quantum algorithm for global optimization.

Indeed, let us select some size $\delta > 0$ (that will be determined later), and let us divide each interval $[\underline{x}_i, \overline{x}_i]$ into $N_i \stackrel{\text{def}}{=} \dfrac{\overline{x}_i - \underline{x}_i}{\delta}$ subintervals of width δ.

This divides the whole box B into into

$$N = N_1 \cdot \ldots \cdot N_n = \prod_{i=1}^{n} \frac{\overline{x}_i - \underline{x}_i}{\delta} = \frac{V}{\delta^n} \tag{4}$$

subboxes, where

$$V \stackrel{\text{def}}{=} (\overline{x}_1 - \underline{x}_1) \cdot \ldots \cdot (\overline{x}_n - \underline{x}_n) \tag{5}$$

is the volume of the original box B.

We can have functions which are 0 everywhere except for one subbox at which this function grows to some value slightly larger than $\varepsilon > 0$ – e.g., equal to $1.1 \cdot \varepsilon$. On this subbox, the function is approximately quadratic. If we have a bound S on the second derivative, we conclude that this function – which starts with 0 at a neighboring subbox – cannot grow faster than $S \cdot x^2$ on this subbox. Thus, to reach a value larger than ε, we need to select δ for which $S \cdot (\delta/2)^2 = 1.1 \cdot \varepsilon$, i.e., the value $\delta \sim \varepsilon^{1/2}$. For this value δ, we get $V/\delta^n \sim \varepsilon^{-(n/2)}$ subboxes.

If we do not explore some values of the optimized function at each of the subboxes, we may miss the subbox that contains the largest value – and thus, we will not be able to localize the point at which the function attains its maximum. Thus, to locate the global maximum, we need at least as many computation steps as there are subboxes – i.e., we need at least time $\sim \varepsilon^{-(n/2)}$.

This lower bound is reachable. Let us show that, similarly to search in unsorted array, this lower bound is reachable: there exists an algorithm that always locates the global maximum in time $\sim \varepsilon^{-(n/2)}$.

First stage of this algorithm: estimation on each subbox. Let us divide the box B into subboxes of linear size δ.

For each such subbox b, since each of its sides has size $\le \delta$, each component x_i of each point $x = (x_1, \ldots, x_n) \in b$ differs from the corresponding component of the subbox's midpoint $x^m \stackrel{\text{def}}{=} (x_1^m, \ldots, x_n^m)$ by no more than $\delta/2$: $|\Delta x_i| \le \delta/2$, where we denoted $\Delta x_i \stackrel{\text{def}}{=} x_i - x_i^m$. Thus, by using known formulas from calculus, we can conclude that for each point $x = (x_1, \ldots, x_n) \in b$, we have

$$f(x_1, \ldots, x_n) = f(x_1^m + \Delta x_1, \ldots, x_n^m + \Delta x_n) =$$

$$f(x_1^m, \ldots, x_n^m) + \sum_{i=1}^{n} c_i \cdot \Delta x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \cdot \Delta x_i \cdot \Delta x_j, \tag{6}$$

where we denoted $c_i \stackrel{\text{def}}{=} \dfrac{\partial f}{\partial x_i}(x_1^m,\ldots,x_n^m)$ and $c_{ij} \stackrel{\text{def}}{=} \dfrac{\partial^2 f}{\partial x_i \partial x_j}(\xi_1,\ldots,\xi_n)$, for some values $(\xi_1,\ldots,\xi_n) \in b$.

We assumed that the function f is twice continuously differentiable. So, all its second derivatives are continuous, and thus, there exists a general bound S on all the values of all second derivatives: $|c_{ij}| \le S$. Because of these bounds, the quadratic terms in the formula (6) are bounded by $n^2 \cdot S \cdot (\delta/2)^2 = O(\delta^2)$.

These estimations lead to a global piece-wise linear approximate function. By considering only linear terms on each subbox, we get an approximate piece-wise linear function $f_{\approx}(x_1,\ldots,x_n)$ which, on each subbox b, has a linear form

$$f_{\approx}(x_1,\ldots,x_n) = f(x_1^m,\ldots,x_n^m) + \sum_{i=1}^{n} c_i \cdot \Delta x_i. \tag{7}$$

For each $x = (x_1,\ldots,x_n)$, we have

$$|f(x_1,\ldots,x_n) - f_{\approx}(x_1,\ldots,x_n)| \le n^2 \cdot S \cdot (\delta/2)^2. \tag{8}$$

Optimizing the approximate function. Let us find the point at which the approximate piece-wise linear function $f_{\approx}(x_1,\ldots,x_n)$ attains its maximum.

For each subbox

$$[x_1^m - \delta/2, x_1^m + \delta/2] \times \ldots \times [x_n^m - \delta/2, x_n^m + \delta/2], \tag{9}$$

as one can easily see:

- the function $f_{\approx}(x_1,\ldots,x_n)$ is increasing with respect to each x_i when $c_i \ge 0$ and
- the function $f_{\approx}(x_1,\ldots,x_n)$ is decreasing with respect to x_i if $c_i \le 0$.

Thus:

- when $c_i \ge 0$, the maximum of the function $f_{\approx}(x_1,\ldots,x_n)$ on this subbox is attained when $x_i = x_i^m + \delta/2$,
- when $c_i \le 0$, the maximum of the function $f_{\approx}(x_1,\ldots,x_n)$ on this subbox is attained when $x_i = x_i^m - \delta/2$.

We can combine both cases by saying that the maximum is attained when $x_i = x_i^m + \text{sign}(c_i) \cdot (\delta/2)$, where $\text{sign}(x)$ is the sign of x (i.e., 1 if $x \ge 0$ and -1 otherwise).

We can repeat this procedure for each subbox, find the corresponding largest value on each subbox, and then find the largest of these values. The point $x^M = (x_1^M,\ldots,x_n^M)$ at which this largest value is attained is thus the point at which the piece-wise linear function $f_{\approx}(x_1,\ldots,x_n)$ attains its maximum.

Proof of correctness. Let us show that the point $x^M = (x_1^M,\ldots,x_n^M)$ is indeed a solution to the given optimization problem.

Indeed, let $x^{\text{opt}} = (x_1^{\text{opt}},\ldots,x_n^{\text{opt}})$ be a point where the original function $f(x_1,\ldots,x_n)$ attains its maximum. Since the approximate function attains its maximum at the

point x^M, its value $f_{\approx}(x^M)$ at this point is larger than or equal to any other value. In particular, we have

$$f_{\approx}(x_1^M,\ldots,x_n^M) \geq f_{\approx}(x_1^{\rm opt},\ldots,x_n^{\rm opt}). \tag{10}$$

Since the functions $f_{\approx}(x_1,\ldots,x_n)$ and $f(x_1,\ldots,x_n)$ are η-close, where we denoted

$$\eta \stackrel{\rm def}{=} n^2\cdot S\cdot(\delta/2)^2, \tag{11}$$

we conclude that

$$f(x_1^M,\ldots,x_n^M) \geq f_{\approx}(x_1^M,\ldots,x_n^M)-\eta \tag{12}$$

and

$$f_{\approx}(x_1^{\rm opt},\ldots,x_n^{\rm opt}) \geq f(x_1^{\rm opt},\ldots,x_n^{\rm opt})-\eta = M-\eta. \tag{13}$$

From (10), (12), and (13), we conclude that

$$f(x_1^M,\ldots,x_n^M) \geq f_{\approx}(x_1^M,\ldots,x_n^M)-\eta \geq f_{\approx}(x_1^{\rm opt},\ldots,x_n^{\rm opt})-\eta \geq$$

$$(M-\eta)-\eta = M-2\eta, \tag{14}$$

i.e., that

$$f(x_1^M,\ldots,x_n^M) \geq M-2\eta. \tag{15}$$

Thus, for $\eta=\varepsilon/2$, we indeed get the solution to the original problem.

Selecting the appropriate value of the parameter δ. So, to solve the original problem with a given ε, we need to select the value δ for which

$$2n^2\cdot S\cdot(\delta/2)^2=\varepsilon, \tag{16}$$

i.e., the value $\delta = c\cdot\varepsilon^{1/2}$, for an appropriate constant c.

How much computation time do we need. In this algorithm, we divide the whole box B of volume V into V/δ^n subboxes of linear size δ. Since $\delta\sim\varepsilon^{1/2}$, the overall number of subboxes is proportional to $\varepsilon^{-n/2}$. On each subbox, the number of computational steps does not depend on ε, so the overall computation time is proportional to the number of boxes, i.e.,

$$T = {\rm const}\cdot\varepsilon^{-n/2}. \tag{17}$$

3 How Quantum Computing Can Help

Preliminary step: bounding the approximate function. Similarly to how we bounded the function on a subbox, we can use the bound S on the second derivative and find the upper bound on the function $f(x_1,\ldots,x_n)$ over the whole box B, i.e., we can find the value $\overline{F}$ for which $f(x_1,\ldots,x_n)\leq\overline{F}$ for all $x=(x_1,\ldots,x_n)\in B$.

This value $\overline{F}$ also serves as the upper bound for the desired maximum M of the function $f(x_1,\ldots,x_n)$. As the lower bound for the maximum, we can take, e.g., the value $\underline{F}$ of the function $f(x_1,\ldots,x_n)$ at the midpoint of the box B. Thus, we know that the maximum M lies in the interval $[\underline{F},\overline{F}]$.

By selecting an appropriate value $\delta \sim \varepsilon^{1/2}$, we can get an approximate function $f_{\approx}(x)$ which is $(\varepsilon/4)$-close to the original function $f(x)$. Because of this closeness, the maximum $M_{\approx}$ of the approximate function is, as we have shown in the previous section, $(\varepsilon/2)$-close to the maximum M and is, thus, located in the interval $[\underline{A}_0,\overline{A}_0]$, where $\underline{A}_0 \stackrel{\text{def}}{=} \underline{F} - \varepsilon/2$ and $\overline{A}_0 \stackrel{\text{def}}{=} \overline{F} + \varepsilon/2$.

Auxiliary quantum algorithm. For each rational value A, we can use Grover's algorithm to find, in time $\sim \sqrt{N}$, one of N subboxes at which the maximum of $f_{\approx}(x)$ on this subbox is larger than or equal to A (or that there is no such subbox).

How we will use the auxiliary quantum algorithm: main idea. Let us assume that we know an interval $[\underline{A},\overline{A}]$ that contains the maximum $M_{\approx}$ of the approximate function. Let us apply the above auxiliary quantum algorithm for $A = (\underline{A}+\overline{A})/2$.

If, applying the auxiliary quantum algorithm, we find our that there is a subbox b for which $f_{\approx}(x_0) \geq A$ for some $x_0 \in b$, then we will be able to conclude that

$$M_{\approx} = \max_{x\in B} f_{\approx}(x) \geq f(x_0) \geq A \tag{18}$$

and thus, that $M_{\approx} \geq A$ and that the actual maximum $M_{\approx}$ of the function $f_{\approx}$ is located somewhere in the interval $[A,\overline{A}]$.

On the other hand, if, after applying the auxiliary quantum algorithm, we find out that no such subbox exists, this means that $f_{\approx}(x) \leq A$ for all x. Thus, the maximum $M_{\approx}$ of the auxiliary function $f_{\approx}(x)$ is also smaller than or equal to A. Hence, in this case, the actual maximum $M_{\approx}$ of the function $f_{\approx}$ is somewhere in the interval $[\underline{A},A]$.

In both cases, we get an interval of half-size that contains the value $M_{\approx}$.

Main algorithm: first part. Now, we can run the following bisection algorithm.

We start with the interval $[\underline{A},\overline{A}] = [\underline{A}_0,\overline{A}_0]$ that contains the actual value $M_{\approx}$.

At each iteration, we apply the above idea with $A = (\underline{A}+\overline{A})/2$, and as a result, we come up with a half-size interval containing $M_{\approx}$.

In k steps, we decrease the width of the interval 2^k times, to $2^{-k}\cdot(\overline{A}-\underline{A})$. In particular, in $k \approx \ln(\varepsilon)$, we can get an interval $[\underline{a},\overline{a}]$ containing $M_{\approx}$ whose width is $\leq \varepsilon/4$: $\underline{a} \leq M_{\approx} \leq \overline{a}$.

Main algorithm: second part. Since the value $\underline{a}$ is smaller than or equal to the maximum $M_{\approx}$ of the approximate function $f_{\approx}(x_1,\ldots,x_n)$, one of the values of this approximate function is indeed greater than or equal to $\underline{a}$.

The above-described auxiliary quantum algorithm will then find, in time $\sim \sqrt{N}$, such a point $x^q = (x_1^q,\ldots,x_n^q)$ for which $f_{\approx}(x_1^q,\ldots,x_n^q) \geq \underline{a}$.

Proof of correctness. Let us prove that the resulting point $x^q = (x_1^q,\ldots,x_n^q)$ indeed solves the original optimization problem.

Indeed, by the very construction of this point, the value $f_{\approx}(x^q)$ is greater than or equal to $\underline{a}$. Since the value $f_{\approx}(x^q)$ cannot exceed the maximum value $M_{\approx}$ of the

approximate function, and this maximum value is $\le \overline{a}$, we conclude that $f_{\approx}(x^q) \le \overline{a}$. Thus, both $f_{\approx}(x^q)$ and $M_{\approx}$ belong to the same interval $[\underline{a}, \overline{a}]$ of width $\le \varepsilon/4$. Thus, the value $f_{\approx}(x^q)$ is $(\varepsilon/4)$-close to the maximum $M_{\approx}$. In particular, this implies that

$$f_{\approx}(x_1^q, \ldots, x_n^q) \ge M_{\approx} - \varepsilon/4. \tag{19}$$

Since the functions $f_{\approx}(x_1, \ldots, x_n)$ and $f(x_1, \ldots, x_n)$ are $(\varepsilon/4)$-close, we can conclude that the maximum values $M_{\approx}$ and M of these two functions are also $(\varepsilon/4)$-close. In particular, this implies that

$$M_{\approx} \ge M - \varepsilon/4. \tag{20}$$

From (19) and (20), can conclude that

$$f(x_1^q, \ldots, x_n^q) \ge M - \varepsilon/2. \tag{21}$$

Since the functions are $(\varepsilon/4)$-close, we conclude that

$$f(x_1^q, \ldots, x_n^q) \ge f_{\approx}(x_1^q, \ldots, x_n^q) - \varepsilon/4 \tag{22}$$

and thus, that

$$f(x_1^q, \ldots, x_n^q) \ge f_{\approx}(x_1^q, \ldots, x_n^q) - \varepsilon/4 \ge (M - \varepsilon/2) - \varepsilon/4 > M - \varepsilon. \tag{23}$$

Thus, we indeed get the desired solution to the optimization problem.

What is the computational complexity of this quantum algorithm. How many computational steps do we need to implement this algorithm?

We need $\sim \ln(\varepsilon)$ iterations each of which requires time

$$\sim \sqrt{N} \sim \sqrt{\varepsilon^{-(n/2)}} = \varepsilon^{-(n/4)}. \tag{24}$$

Thus, the overall computation time T_q of this quantum algorithm is equal to

$$T_q \sim \varepsilon^{-(n/4)} \cdot \ln(\varepsilon). \tag{25}$$

How faster is this quantum algorithm than the non-quantum optimization? We know that the computation time T of the non-quantum algorithm is $T \sim \varepsilon^{-(n/2)}$; thus, $\varepsilon^{-(n/4)} \sim \sqrt{T}$.

Here, $\varepsilon \sim T^{-(2/n)}$, and thus, $\ln(\varepsilon) \sim \ln(T)$. Thus, we conclude that

$$T_q \sim \sqrt{T} \cdot \ln(T). \tag{26}$$

The main result is thus proven.

Acknowledgments

This work was supported in part by the US National Science Foundation grant HRD-1242122 (Cyber-ShARE Center of Excellence).

The authors are thankful for all the participants of the NMSU/UTEP Workshop on Mathematics, Computer Science, and Computational Science (Las Cruces, New Mexico, April 6, 2019) for valuable suggestions.

References

1. L. K. Grover, "A fast quantum mechanical algorithm for database search", *Proceedings of the 28th ACM Symposium on Theory of Computing*, 1996, pp. 212–219.
2. L. K. Grover, "Quantum mechanics helps in searching for a needle in a haystack", *Physical Reviews Letters*, 1997, Vol. 79, No. 2, pp. 325–328.
3. M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, 2000.