5

Continuous optimization

Continuous optimization problems arise throughout science and industry. On their face, continuous optimization problems rarely seem quantum mechanical; nevertheless, quantum algorithms have been proposed for accelerating both convex and nonconvex continuous optimization. Most of the research on these algorithms thus far has been to develop and utilize the diverse set of primitive ingredients that give rise to potential quantum advantage in this space, without an eye toward the end-to-end practicality of the algorithms. Developing a better understanding of the practicality of these approaches should be a focus of future work.

We refer the reader to [6] for a comprehensive survey of quantum methods for continuous and combinatorial optimization.

The authors are grateful to Sander Gribling for reviewing this chapter.

5.1 Zero-sum games: Computing Nash equilibria

Overview

In a two-player zero-sum game, each player independently chooses an action and then receives a "payoff" (such that the sum of the payoffs is always zero) that depends on which pair of actions was chosen. A Nash equilibrium is an optimal way of (probabilistically) choosing an action that maximizes a player's worst-case payoff. The problem of computing a Nash equilibrium is, in a certain sense, equivalent to solving a linear program (LP): computing a Nash equilibrium is a special case of LP, and conversely any LP can be reduced to computing a Nash equilibrium at the expense of introducing dependencies on a certain instance-specific "scale-invariant" precision parameter [46]. However, the quantum approach to solving LPs based on the multiplicative weights update method [46] is more efficient in the special case of computing Nash equilibria, and has fewer caveats—notably, it avoids the dependence on the difficult-to-predict scale-invariant precision parameter. It gives a potentially quadratic speedup over its classical counterpart.

Actual end-to-end problem(s) solved

A two-player zero-sum game is defined by an $n \times m$ matrix A called the "payoff matrix," which specifies how much player 1 wins from player 2 when player 1 chooses action $i \in [n]$ and player 2 chooses action $j \in [m]$. A pure strategy is one in which the players deterministically choose one fixed action $i \in [n]$ (or $j \in [m]$) in each game. By contrast, a mixed strategy is one in which players randomly choose an action, according to some probability distribution. Assume the entries of A are between -1 and 1. A Nash equilibrium is an optimal (generally mixed) strategy that maximizes a player's worst-case payoff regardless of the other player's choice. That is, a distribution $y \in \Delta^m$, where Δ^m denotes the m-dimensional probability simplex, is an optimal strategy for player 2 if it is the argument that optimizes the equation

$$\lambda^* = \min_{y \in \Delta^m} \max_{i \in [n]} e_i^\top A y,$$

where $[n]$ denotes the set of actions available to player 1, and e_i denotes a basis state associated with action i. The quantity λ^* is the value of the game. This can be rewritten explicitly [46] as the following LP

$$\begin{aligned} & \min_{y \in \mathbb{R}^m} \lambda \\ \text{subject to} \qquad & Ay \le \lambda \mathbf{1}, \qquad \sum_j y_j = 1, \qquad y_j \ge 0 \ \forall j, \end{aligned}$$

where $\mathbf{1}$ is the all-ones vector. The dual LP for the above then corresponds to computing the Nash equilibrium for player 1.

The end-to-end problem solved is to, given access to the entries of the matrix A and an error parameter ϵ, compute a probability vector y such that

$$Ay \le (\lambda^* + \epsilon)\mathbf{1}\,.$$

Dominant resource cost/complexity

The quantum algorithm builds on a classical algorithm based on the multiplicative weights update method from [459]. With probability at least $1-\delta$, the classical algorithm finds a solution y that approximates a Nash equilibrium to error ϵ after $\lceil 16 \ln(nm/\delta)/\epsilon^2 \rceil$ iterations, where the cost per iteration is $n+m$ queries to the entries of the matrix A and $O(n+m)$ other arithmetic operations

[46, Lemma 3]. An important subroutine of each iteration is a Gibbs sampling step for a diagonal matrix (a special case of the general quantum Gibbs sampling problem in which any Hermitian matrix is allowable). When the matrix A is sparse, the number of queries per iteration can be reduced to $2s$, where s is the maximum number of nonzero entries in a row or column of A, and the total time per iteration can be reduced to $\widetilde{O}(s)$ [46, Lemma 4].

The quantum algorithm assumes coherent access to the matrix entries of A. Through amplitude amplification and the related subroutines of amplitude estimation and minimum finding, the quantum algorithm of [46] speeds up the Gibbs sampling task and reduces the maximum cost of an iteration to $\widetilde{O}(\sqrt{n+m}/\epsilon)$ queries to the matrix elements of A and an equal amount of time complexity, where $\widetilde{O}$ notation suppresses logarithmic factors. In the case that the matrices are sparse, the maximum cost of an iteration is reduced to $\widetilde{O}(\sqrt{s}/\epsilon^{1.5})$. The work of [178] introduces a technique called dynamic Gibbs sampling, which exploits the fact that the distribution to be sampled changes slowly from iteration to iteration and further reduces the iteration cost to $\widetilde{O}(\sqrt{n+m}/\epsilon^{1/2} + 1/\epsilon)$ in the dense case. This gives a total query and time complexity roughly given by

$$\text{dense:}\quad \left(\frac{16\ln(nm)}{\epsilon^2}\text{ iters.}\right)\times\left(\widetilde{O}\left(\frac{\sqrt{n+m}}{\sqrt{\epsilon}}+\frac{1}{\epsilon}\right)\text{per iter.}\right)=\widetilde{O}\left(\frac{\sqrt{n+m}}{\epsilon^{2.5}}+\frac{1}{\epsilon^3}\right)$$

$$\text{sparse:}\quad \left(\frac{16\ln(nm)}{\epsilon^2}\text{ iters.}\right)\times\left(\widetilde{O}\left(\frac{\sqrt{s}}{\epsilon^{1.5}}\right)\text{per iter.}\right)=\widetilde{O}\left(\frac{\sqrt{s}}{\epsilon^{3.5}}\right).$$

This complexity assumes access to a quantum random access memory (QRAM). Without a QRAM, the cost per iteration increases by a factor $\widetilde{O}(1/\epsilon^2)$.

See also [680], which independently from [46] gave a quantum algorithm that solves zero-sum games with slightly worse ϵ dependence, as well as [681], which gave quantum algorithms for generalizations of zero-sum games to other vector norms.

Existing resource estimates

There are no existing explicit resource estimates for this algorithm.

Caveats

- Due to poor dependence of the complexity on the error ϵ, this algorithm is only likely to be useful in situations where it is not necessary to learn the optimal strategy to high precision. It is unclear when such situations arise in practice.

- As mentioned above, if no QRAM is available, the runtime suffers a $\widetilde{O}(1/\epsilon^2)$ time slowdown.
- A fully end-to-end analysis should also consider the exact way that the queries to the matrix entries of A are implemented. If they are given in a classical database, a large $O(nm)$-size QRAM may also be required to implement the queries in polylog(mn) time. Note that this would be separate from the $\widetilde{O}(1/\epsilon^2)$-size QRAM the algorithm uses to reduce the time complexity. To avoid the QRAM requirement for implementing a query, it must be the case that the matrix entries are efficiently computable in some other way.

Comparable classical complexity and challenging instance sizes

The classical version of the quantum algorithm has time and query complexity given by [46, Section 2]

$$\text{dense:} \quad \left(\frac{16\ln(nm)}{\epsilon^2}\text{ iters.}\right) \times \left(O\left(n+m\right)\text{ per iter.}\right) = \widetilde{O}\left(\frac{n+m}{\epsilon^2}\right)$$

$$\text{sparse:} \quad \left(\frac{16\ln(nm)}{\epsilon^2}\text{ iters.}\right) \times \left(\widetilde{O}\left(s\right)\text{ per iter.}\right) = \widetilde{O}\left(\frac{s}{\epsilon^2}\right).$$

Alternatively, the problem could be solved using other approaches for solving the associated LP. Classical interior point methods for LPs can achieve $O(n^\omega \log(1/\epsilon))$ runtime in the common case that $m = O(n)$ [304], where $\omega < 2.37$ is the matrix multiplication exponent. This runtime exhibits better ϵ dependence at the expense of worse n dependence. Note that quantum interior point methods have also been proposed for conic programs like LPs, but whether they could yield a speedup over classical interior point methods would depend on the scaling of certain instance-specific parameters.

Speedup

The quantum complexity has a quadratic improvement in complexity with respect to the parameter $n + m$, and a polynomial slowdown with respect to the parameter ϵ.

Outlook

It is difficult to assess whether a practical advantage could be obtained in the setting of zero-sum games without further investigation of how queries to matrix elements are accomplished, an assessment of constant prefactors involved in the algorithm, and consideration of any additional overheads from fault-tolerant quantum computation. The theoretical speedup available is quadratic and may require a medium- or large-scale QRAM. This speedup may not be sufficiently large to overcome these overheads in practice.

It is perhaps instructive to compare the outlook of zero-sum games to conic programming more generally. On the one hand, unlike the algorithm for general SDPs and LPs, the algorithm for zero-sum games does not have a complexity dependence on instance-specific parameters denoting the size of the primal and dual solutions. This makes it easier to evaluate the runtime of the algorithm and more likely that it can be an effective algorithm. On the other hand, a core subroutine of the quantum algorithm is to perform *classical* Gibbs sampling quadratically faster than a classical computer can using techniques like amplitude amplification. However, it is not clear how the speedup could be made greater than quadratic, even in special cases. A similar subroutine is required in the multiplicative weights approach to solving SDPs, but in that case, the Gibbs state to be sampled is a truly quantum state (i.e., nondiagonal in the computational basis), rather than a classical state. Using more advanced methods for Gibbs sampling, it is possible that in some special cases there could be a superquadratic quantum speedup for SDPs that would not be available for the simpler case of LPs and zero-sum games.

5.2 Conic programming: Solving LPs, SOCPs, and SDPs

Overview

Conic programs are a specific subclass of convex optimization problems, where the objective function is linear and the convex constraints are restrictions to the intersection of affine spaces and certain cones within $\mathbb{R}^n$. Commonly considered cones are the positive orthant, the second-order cone ("ice-cream cone"), and the semidefinite cone, which give rise to linear programs (LPs), second-order cone programs (SOCPs), and semidefinite programs (SDPs), respectively. This framework remains quite general, and many real-world problems can be reduced to a conic program. However, the additional structure of the program allows for more efficient classical and quantum algorithms, compared to completely general convex problems.

Algorithms for LPs, SOCPs, and SDPs have long been a topic of study. Today, the best classical algorithms are based on interior point methods (IPMs) [304, 778, 538] and cutting-plane methods [671, 575], but other algorithms based on the multiplicative weights update (MWU) method [57, 56, 58] exist and can be superior in a regime where high precision is not required. Both of these approaches can be turned into quantum algorithms with potential to deliver asymptotic quantum speedup for general LPs, SOCPs, and SDPs. However, the runtime of the quantum algorithm typically depends on additional instance-specific parameters, which makes it difficult to produce a general apples-to-apples comparison with classical algorithms.

Actual end-to-end problem(s) solved

- Linear programs (LPs) are the simplest convex program. An LP instance is specified by an $m \times n$ matrix A, an n-dimensional vector c, and an m-dimensional vector b. The problem can then be written as

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} \langle c, x \rangle \\ \text{subject to } &Ax = b \\ &x_i \geq 0 \text{ for } i = 1, \ldots, n\,, \end{aligned}$$

  where notation $\langle u, v \rangle$ denotes the standard dot product of vectors u and v. The function $\langle c, x \rangle$, which is linear in x, is called the objective function, and a point x is called feasible if it satisfies the linear equality[1] constraints $Ax = b$ as well as the positivity constraints $x_i \geq 0$ for all i. We denote the feasible point that optimizes the objective function by x^*. Let ϵ be a precision parameter. The actual end-to-end problem solved is to take as input a classical description of the problem instance (c, A, b, ϵ) and output a classical description of a feasible point x for which $\langle c, x \rangle \leq \langle c, x^* \rangle + \epsilon$. The set of points that obey the positivity constraints $x_i \geq 0$ forms the positive orthant of the vector space $\mathbb{R}^n$. This set meets the mathematical definition of a convex cone: for any points u and v in the set and any non-negative scalars $\alpha, \beta \geq 0$, the point $\alpha u + \beta v$ is also in the set.
- Second-order cone programs (SOCPs) are formed by replacing the positivity constraints in the definition of LPs with one or more second-order cone constraints, where the second-order cone of dimension k is defined to include points $(x_0; x_1; \ldots; x_{k-1}) \in \mathbb{R}^k$ for which $x_0^2 \geq x_1^2 + \cdots + x_{k-1}^2$.
- Semidefinite programs (SDPs) are formed by replacing the n-dimensional vector x in the definition of LPs with an $n \times n$ symmetric matrix X and replacing the positive orthant constraint with the conic constraint that X is a positive semidefinite matrix. Denote the set of $n \times n$ symmetric matrices by $\mathbb{S}^n$, and for any pair of matrices $U, V \in \mathbb{S}^n$, define the notation $\langle U, V \rangle = \mathrm{tr}(UV)$ (which generalizes the standard dot product). Then, an SDP instance is specified by matrices $C, A^{(1)}, A^{(2)}, \ldots, A^{(m)} \in \mathbb{S}^n$, as well as $b \in \mathbb{R}^m$, and

[1] Inequality constraints of the form $Ax \leq b$ can be converted to linear equality constraints and positivity constraints by introducing a vector of slack variables s and imposing $Ax + s = b$ and $s_i \geq 0$ for all i. An analogous trick is possible for SOCP and SDP.

can be written as

$$\begin{aligned} &\min_{X \in \mathbb{S}^n} \langle C, X \rangle \\ \text{subject to } &\langle A^{(j)}, X \rangle = b_j \text{ for } j = 1, \ldots, m \\ &X \succeq 0 \,, \end{aligned}$$

where $X \succeq 0$ denotes the constraint that X is positive semidefinite.

In the LP or SDP case, we might also require as input parameters R and r, where R is a known upper bound on the size of the solution in the sense that $\sum_i |x_i| \leq R$ (LP) or $\mathrm{tr}(X) \leq R$ (SDP), and where r is an analogous upper bound on the size of the solution to the *dual* program (not written explicitly here, see [181, 48, 45]).

Dominant resource cost/complexity

Two separate approaches to solving conic programs with quantum algorithms have been proposed in the literature. Both methods start with classical algorithms and replace some of the subroutines with quantum algorithms.

(i) Quantum interior point methods (QIPMs) for LPs [610], SOCPs [612, 68], and SDPs [610, 70, 537] have been proposed. In the standard approach, these methods start with classical interior point methods, for which the core step is solving a linear system, and simply replace the classical linear system solver with a quantum linear system solver (QLSS), combined with pure state quantum tomography. Given a linear system $Gu = v$, the QLSS produces a quantum state $|u\rangle$, and quantum tomography is subsequently used to gain a classical estimate of the amplitudes of $|u\rangle$ in the computational basis. The QLSS ingredient introduces complexity dependence on a parameter $\kappa = \|G\| \, \|G^{-1}\|$, the condition number of G, where $\|\cdot\|$ denotes the spectral norm. Additionally, the QLSS requires that the classical data defining G be loaded in the form of a block-encoding, for which the standard construction introduces a dependence on the factor $\zeta = \|G\|_F \|G\|^{-1}$, where $\|\cdot\|_F$ denotes the Frobenius norm. Finally, the tomography ingredient introduces a complexity dependence on a parameter ξ, defined as the precision to which the vector u must be classically learned, measured in ℓ_2 norm. Assuming m is on the order of the number of degrees of freedom (i.e., $O(n)$ in the case of LP and SOCP, and $O(n^2)$ in the case of SDP), the number of queries the QIPM makes to block-encodings of the input matrices is

$$\text{LP, SOCP [68]:} \qquad \widetilde{O}\left(\frac{n^{1.5}\zeta\kappa}{\xi}\log(1/\epsilon)\right)$$
$$\text{SDP [610, 70]:} \qquad \widetilde{O}\left(\frac{n^{2.5}\zeta\kappa}{\xi}\log(1/\epsilon)\right),$$

where the $\widetilde{O}$ notation hides logarithmic factors. Note that depending on how ξ is defined, extra factors of κ may be required. Moreover, note that the complexity statements in [70] go further and analyze the worst-case dependence of ξ on the overall error ϵ, and additionally make the worst-case replacement $\zeta \leq O(n)$—this explains the deviation in our presentation from the bounds in [70]. We do not include these worst-case assumptions on ζ and ξ because it is possible they are not achieved in practice.[2] Generally speaking, the numerical values of κ, ζ, and ξ are not possible to determine in advance for a specific application; empirical investigations at small system sizes such as those of [611, 328] require an assumption that trends observed accurately extrapolate to other untested instances and to larger system sizes. The block-encoding queries can be executed in circuit depth polylog$(n+m, 1/\epsilon)$, which can also be absorbed into the $\widetilde{O}$ notation (although it is important to note that the circuit *size* is generally $O(n^2)$)—this is equivalent to an assumption of log-depth quantum random access memory (QRAM). If the input matrices are sparse or given in a form other than as a list of matrix entries, there may be other more efficient methods for block-encoding; in this case the parameter ζ might be replaced with another parameter $\alpha > 1$, whose value would depend on the block-encoding method. It should also be noted that in addition to the quantum complexity quoted above, the QIPM can require (depending on the precise formulation of the QIPM) purely classical complexity on the order of $O(n^{2.5})$ for LP/SOCP and $O(n^{4.5})$ for SDP.

Alternatives to the standard approach above have been proposed. For "tall" LPs (where $m \gg n$), one can quantize interior point methods in a distinct way that avoids the QLSS and dependence on any condition numbers. Specifically, [51] gave an algorithm that runs in time $\widetilde{O}(\sqrt{m}\log(1/\epsilon)) \cdot \text{poly}(n)$. The algorithm leverages primitives for spectral approximation (i.e., given a tall matrix B, finding a smaller matrix

[2] For example, numerical results in [611] for small instances of an SOCP formulation of the portfolio optimization problem suggested that the ζ parameter was upper bounded by a small constant, and similar numerical investigations in [328] suggest that ξ can be independent of the target error ϵ, at least for the instances that were simulated.

$\tilde{B}$ for which $\tilde{B}^\intercal \tilde{B} \approx B^\intercal B$), and approximate matrix-vector multiplication, as well as multivariate mean estimation [310] (which is related to quantum gradient estimation).

(ii) Quantum algorithms based on the multiplicative weights update (MWU) method have been proposed for SDP [181, 182, 48, 45] and LP [48, 46]. The quantum algorithm closely follows the classical algorithm based on MWU to iteratively update a candidate solution to the program. Each iteration is carried out using quantum subroutines, including Gibbs sampling, as well as Grover search and quantum minimum finding [367, 48] (a direct application of Grover search). Let s denote the sparsity, that is, the maximum number of nonzero entries in any row or column of the matrices composing the problem input (thus, $s \leq \max(m, n)$). Then, the number of queries the algorithm makes to the matrix entries (assuming a sparse access input model) has been upper bounded by

$$\text{LP [178]:} \qquad \widetilde{O}\left(\sqrt{s}\left(\frac{rR}{\epsilon}\right)^{3.5}\right)$$

$$\text{SDP [45]:} \qquad \widetilde{O}\left(s\sqrt{m}\left(\frac{rR}{\epsilon}\right)^{4} + s\sqrt{n}\left(\frac{rR}{\epsilon}\right)^{5}\right),$$

where r, R are the parameters related to the size of the primal and dual solutions, defined above. The sparse access queries can be implemented with quantum circuits of size polylog(m, n) if the matrix entries are given by succinct formulas computable in polylog(n, m) time. Otherwise, their implementation can be accomplished with circuits of polylog(m, n) depth (but poly(m, n) size) assuming availability of log-depth QRAM. In [45], the input model was generalized to a "quantum operator input model," based on block-encodings where s is replaced by the block-encoding normalization factor α in the runtime expressions, but here again the full end-to-end complexity must account for the gate cost of implementing the block-encoding. Note that it is possible the ϵ-dependence of the runtime for LP could be slightly improved by applying the dynamic Gibbs sampling method of [178] together with the reduction from LP to zero-sum games in [46].

The runtime expressions for the QIPM approach and the MWU approach are not directly comparable, as the former depends on instance-specific parameters κ, ζ, and ξ, while the latter depends on instance-specific parameters r and R. However, note that the explicit n-dependence is better in the case of MWU than QIPM, while the ϵ-dependence is worse.

Existing resource estimates

Neither of the approaches for conic programs have garnered study at the level of resource estimates for physical devices. Reference [328] performed a resource analysis for a QIPM at the logical level, but did not analyze additional overheads due to error correction. The goal of that analysis was to completely compile the QIPM for SOCP into Clifford gates and T gates, and then to numerically estimate the parameters κ, ζ, and ξ for the particular use case of financial portfolio optimization, which can be reduced to SOCP. A salient feature of the QIPM is that $\mathcal{O}(n+m) \times \mathcal{O}(n+m)$ matrices of classical data must be repeatedly accessed by the QLSS via block-encoding, necessitating a large-scale QRAM with $\mathcal{O}(n^2)$ qubits. Accordingly, for SOCPs with $n = 500$ and $m = 400$ (which are still easily solved on classical computers) it was estimated that 8 million logical qubits would be needed. The total number of T gates needed for the same instance size was on the order of 10^{29}, which can be distributed over roughly 10^{24} layers. These estimates would likely be improved by incorporating subsequent improvements to the underlying primitives of tomography [49] and QLSS [571, 327].

We are not aware of an analogous logical resource analysis for the MWU approach to conic programming. Such an analysis would be valuable and should ideally choose a specific use case to be able to evaluate the size of all parameters involved. A use case that may fit this criteria is solving the SDP relaxation of binary quadratic optimization problems, where r and R can be bounded; quantum algorithms for this task have been studied in [183, 71].

Caveats

- The QIPM approach requires a large-scale QRAM of size $\mathcal{O}(n^2)$. This is a necessary ingredient to retain any hope of a speedup, and for relevant choice of n the associated hardware requirements could be prohibitively large. Note that recent work of [69] gave a method for solving LPs that, like QIPMs, follows the central path to the optimal point, but does so in a distinct, non-iterative way, and has the potential for a small polynomial speedup without the need for a QRAM.
- The standard QIPM approach has a weak case for a large asymptotic speedup: even under optimal circumstances, the asymptotic speedup over classical interior point methods is less than quadratic. See Chapter 22 on the QIPM approach for more information.
- The MWU approach also requires a large-scale QRAM of size $\mathcal{O}(\min(ms, ns))$ to implement the queries to the arbitrary entries of the s-sparse input matrices. This could be avoided if the matrix elements are efficiently computable.

- Beyond the number of queries to the input data (and the cost to implement those queries), the MWU approach for LP and SDP requires some additional complexity deriving from a step in the algorithm where it prepares a state with $\mathcal{O}(R^2r^2/\epsilon^2)$ nonzero amplitudes stored in a classical database. The cost of state preparation from classical data is $\widetilde{\mathcal{O}}(R^2r^2/\epsilon^2)$ total gates, which can be parallelized to circuit depth $\mathrm{polylog}(R^2r^2/\epsilon^2)$. If the cost of state preparation is taken to be equal to the circuit depth (similar to the assumption of access to a medium-scale QRAM), the additional complexity is on the same order as the number of queries. If the cost is taken to be equal to the circuit size, the additional complexity is a factor $\widetilde{\mathcal{O}}(R^2r^2/\epsilon^2)$ larger than the query complexity quoted above.
- The MWU approach has poor dependence on error ϵ; for SDPs it is ϵ^{-5}. Even at modest choices of ϵ, this may lead the algorithm to be impractical pending significant improvements.
- A general caveat that applies to both approaches is that the appearance of instance-specific parameters makes it difficult to predict the performance of these algorithms for more specific applications.

Comparable classical complexity and challenging instance sizes

As in the quantum case, there are multiple distinct approaches in the classical case.

(i) Classical interior point methods (CIPMs): There exist fast IPM-based software implementations for solving conic programs, such as ECOS [355], MOSEK [38], Gurobi [470], SCIP [164], and CPLEX.[3] These solvers can solve instances with thousands of variables in a matter of seconds on a standard laptop (e.g., [355]). However, the runtime scaling is poor and scaling too far beyond this regime leads the solvers to be far less practical. Many variants of IPMs exist; the runtime of the best provably correct classical IPMs for the regime where the number of constraints is roughly equal to the number of degrees of freedom is

$$\begin{aligned} \text{LP [304]:} \quad & \widetilde{\mathcal{O}}(n^{\omega}\log(1/\epsilon)) \\ \text{SOCP [778]:} \quad & \widetilde{\mathcal{O}}\left(n^{\omega+0.5}\log(1/\epsilon)\right) \\ \text{SDP [538]:} \quad & \widetilde{\mathcal{O}}\left(n^{2\omega}\log(1/\epsilon)\right), \end{aligned}$$

[3] In practice, these solvers are not solely based on IPMs and utilize many methods at once. Additionally, they employ heuristic preprocessing methods to transform and simplify inputs prior to applying an IPM. Note that they are useful also for nonconvex problems such as mixed-integer programs, where LP-solving can often be an important subroutine.

where $\omega < 2.37$ is the matrix multiplication exponent. It is plausible that, with some attention, the extra $n^{0.5}$ factor for SOCP could be eliminated with modern techniques. Additionally, the runtime can be somewhat reduced when the number of constraints is much less than the number of degrees of freedom; for example, the n-dependence of the complexity of the CIPM for SDP in [574] can be as low as $\widetilde{O}(n^{2.5})$ when there are few constraints. On practical instances, employing techniques for fast matrix multiplication is often not beneficial, and Gaussian elimination–like methods are used, where n^{ω} is replaced with n^3. Note that, alternatively, by using iterative classical linear system solvers, such as the randomized Kaczmarz method [959], each n^{ω} factor could be replaced by a factor of n at the cost of a linear dependence on $(\kappa\zeta)^2$, which could be superior if the matrices are well conditioned.

For tall LPs ($m \gg n$), CIPMs can achieve scaling nearly linear in the number of matrix entries: the algorithm of [185] runs in time $O(mn+n^3)$.

We refer the reader to [1053, 1052, 797, 438] for additional information on CIPMs and their historical development.

(ii) Classical MWU methods: A classical complexity statement for LPs is inferred from the reduction in [46] from LPs to zero-sum games and the classical analysis that appears there. For the SDP case, references in the classical literature appear only to examine specific subclasses of SDPs (e.g., [57, 56]). A general statement of the classical complexity for SDPs appears alongside the quantum algorithm in [48, Section 2.4]:

$$\text{LP [46]:} \qquad \widetilde{O}\left(s\left(\frac{rR}{\epsilon}\right)^{3.5}\right)$$

$$\text{SDP [48]:} \qquad \widetilde{O}\left(snm\left(\frac{rR}{\epsilon}\right)^{4} + sn\left(\frac{rR}{\epsilon}\right)^{7}\right).$$

(iii) Cutting-plane methods: These classical methods are used for SDPs and can outperform IPMs when the number of constraints is small. The best algorithm, based on [671, 575], has runtime $O(m(mn^2 + n^{\omega} + m^2)\log(1/\epsilon))$, which can be as low as $O(n^{\omega})$ when m is small.

It is important to note that the algorithms with the best provable complexities may not be the ones that are most useful in practice.

Speedup

For both the IPM approach and the MWU approach, there can be at most a polynomial quantum speedup: upper and lower bounds scaling polynomially with n are known in both the classical and quantum cases [45]. The speedup of the QIPM method depends on the scaling of κ with n, but the speedup cannot be more than quadratic. For the MWU method, if $m = O(n)$, $s = O(1)$, and $rR/\epsilon = O(1)$, existing bounds on the classical and quantum complexities leave open the possibility of a quartic $\widetilde{O}(n^2) \to \widetilde{O}(\sqrt{n})$ speedup. However, it is unclear if the classical complexity quoted in [48, Section 2.4] is optimal (e.g., if the classical $\widetilde{O}(mn)$ scaling could be improved to $\widetilde{O}(m+n)$, the available quantum speedup would be at most quadratic). There is a possibility that the speedup could be larger in practice if the Gibbs sampling routine is faster on actual instances than its worst-case upper bounds suggest, perhaps by utilizing Monte Carlo–style approaches to Gibbs sampling.

Outlook

It is very plausible that an asymptotic polynomial speedup can be obtained in problem size using the MWU method for solving LPs or SDPs, but the speedup appears only quadratic, and an assessment of practicality depends on the scaling of certain unspecified instance-specific parameters. Similarly, the standard QIPM method could bring a subquadratic speedup but only under certain assumptions about the condition number of certain matrices. The alternative QIPM method of [51] could deliver a nearly quadratic speedup without assumptions on the condition number in the case the LP constraint matrix is very tall. In any case, these quadratic and subquadratic speedups alone might be regarded as unlikely to yield practical speedups after error correction overheads and slower quantum clock speeds are considered. Future work should aim to find additional asymptotic speedups while focusing on specific practically relevant use cases that allow the unspecified parameters to be evaluated.

5.3 General convex optimization

Overview

A convex problem asks to minimize a convex function f over a convex set K, where K is a subset of $\mathbb{R}^n$. Here we examine the situation where the value of $f(x)$ and the membership of x in the set K can each be efficiently computed classically. However, we do not exploit/assume any additional structure that may be present in f or K. This situation contrasts with that of solving conic programs, where f is linear and K is an intersection of convex cones and affine spaces, features that can be exploited to yield more efficient classical and quantum algorithms.

A so-called “zeroth-order” solution to this problem solves it simply by adaptively evaluating $f(x)$ and $x \in K$ for different values of x. For the zeroth-order approach, a quantum algorithm can obtain a quadratic speedup with respect to the number of times these functions are evaluated, reducing it from $\widetilde{O}(n^2)$ to $\widetilde{O}(n)$, where $\widetilde{O}$ notation hides factors polylogarithmic in n and other parameters. This could lead to a practical speedup only if the cost to evaluate $f(x)$ and $x \in K$ is large, and lack of structure rules out other, possibly faster, approaches to solving the problem.

Actual end-to-end problem(s) solved

Suppose we have classical algorithms $\mathcal{A}_f$ for computing $f(x)$ and $\mathcal{A}_K$ for computing $x \in K$ (“membership oracle”), which require C_f and C_K gates to perform with a reversible classical circuit, respectively. Suppose further we have an initial point $x_0 \in K$ and that we have two numbers r and R for which we know that $B(x_0, r) \subset K \subset B(x_0, R)$, where $B(y,t) = \{z \in \mathbb{R}^n : \|z - y\| \leq t\}$ denotes the ball of radius t centered at y. Using $\mathcal{A}_f$, $\mathcal{A}_K$, x_0, r, R, and ϵ as input, the output is a point $\tilde{x} \in K$ that is ϵ-optimal, that is, it satisfies

$$f(\tilde{x}) \leq \min_{x \in K} f(x) + \epsilon \,.$$

Dominant resource cost/complexity

The work of [245] and [47] independently establish that there is a quantum algorithm that solves this problem with gate complexity upper bounded by

$$\left[(C_f + C_K)n + n^3\right] \cdot \mathrm{polylog}(nR/r\epsilon)\,,$$

where the polylogarithmic factors were left unspecified. The rough idea behind the algorithm is to leverage the quantum gradient estimation algorithm to implement a *separation oracle*—a routine that determines membership $x \in K$ and when $x \notin K$ outputs a hyperplane separating x from all points in K—using only $\widetilde{O}(1)$ queries to algorithm $\mathcal{A}_K$ and $\mathcal{A}_f$. It had been previously established that $\widetilde{O}(n)$ queries to a separation oracle then suffice to perform optimization [671], where $\widetilde{O}$ denotes that logarithmic factors have been suppressed.

Existing resource estimates

There have not been any explicit resource estimates for this algorithm. It may not make sense to perform such an estimate without a more concrete scenario in mind, as the estimate would highly depend on the complexity of performing the circuits for $\mathcal{A}_f$ and $\mathcal{A}_K$. The estimate would also require a more detailed accounting of the hidden polylogarithmic factors in the complexity statements above, and it would only be meaningful if the comparable classical complexity for solving the same problem using the best available algorithm were well understood.

Caveats

One caveat is that the quantum algorithm must coherently perform reversible implementations of the classical functions that compute $f(x)$ and $x \in K$. Compared to a nonreversible classical implementation, this may cost additional ancilla qubits and gates. Another caveat relates to the scenario where $f(x)$ and $x \in K$ are determined by classical data stored in a classical database. Such a situation may appear to be an appealing place to look for applications of this algorithm because when f and K are determined empirically rather than analytically, it becomes easier to argue that there is no structure that can be exploited. However, in such a situation, implementing $\mathcal{A}_f$ and $\mathcal{A}_K$ would require a large gate complexity, so C_f and C_K would scale with the size of the classical database. It would almost certainly be the case that a quantum random access memory (QRAM) admitting log-depth queries would be needed in order for the algorithm to remain competitive with classical implementations that have access to classical RAM, and the practical feasibility of building a large-scale log-depth QRAM has many additional caveats.

Another caveat is that there may not be many practical situations that are compatible with a quantum speedup by this algorithm. The source of the speedup in [245, 47] comes from a separation between the complexity of computing the gradient of f classically vs. quantumly using calls to the function f. Classically, this requires at least linear-in-n number of calls. Quantumly, it can be done in $\tilde{O}(1)$ calls using the quantum algorithm for gradient estimation. In both the classical and the quantum case, the gradient can subsequently be used to construct a "separation" oracle for the set K, which is then used to solve the convex problem.

Thus, a speedup is only possible if there is no obvious way to classically compute the gradient of f other than to evaluate f at many points. This criterion is violated in many practical situations, which are often said to obey a "cheap gradient principle" [457, 163] that asserts that the gradient of f can be computed in time comparable to the time required to evaluate f. For example, the fact that gradients are cheap is crucial for training modern machine learning models with a large number of parameters. When this is the case, the algorithms from [245, 47] do not offer a speedup. On the other hand, as observed in [47, Footnote 19] a nontrivial example of a problem where the cheap gradient principle may fail (enabling a possible advantage for these quantum algorithms) is the moment polytope problem, which has connections to quantum information [211].

When both the function f and the gradient of f can be evaluated at unit cost, this constitutes "first-order" optimization, which can be solved classically by gradient descent. However, gradient descent does not generally offer a quantum speedup, as general quantum lower bounds match classical upper bounds for first-order optimization, although a quantum speedup could exist in specific cases [410]. Indeed, for any p, there is no general quantum speedup for pth-order optimization, that is, the setting where an oracle provides access to the function and its first p derivatives [409]. For a comprehensive exposition of classical methods for black-box optimization, see [798].

Comparable classical complexity

The best classical algorithm [672] in the same setting has complexity

$$\left[(C'_f + C'_K)n^2 + n^3\right] \cdot \mathrm{polylog}(nR/r\epsilon)\,,$$

where C'_f and C'_K denote the classical complexity of evaluating f and querying membership in K, respectively, without the restriction that the circuit be reversible.

Speedup

The speedup is greatest when quantities C_f and C_K are large compared to n and roughly equal to C'_f and C'_K. In this case, the quantum algorithm can provide an $\tilde{O}(n)$ speedup, which is at best a polynomial speedup. The maximal power of the polynomial would be obtained if $C_f + C_K \approx C'_f + C'_K$ scales as n^2, corresponding to a subquadratic speedup from $\tilde{O}(n^4)$ to $\tilde{O}(n^3)$.

Outlook

The only analyses of this strategy are theoretical in nature, interested more so in the query complexity of solving this problem than any specific applications it might have. As such, the analysis is not sufficiently fine-grained to determine any impact from constant prefactors or logarithmic prefactors. While a quadratic speedup in query complexity is possible, the maximal speedup in gate complexity is smaller than quadratic. Moreover, there is a lack of concrete problems that fit into the paradigm of "structureless" quantum convex optimization. Together, these factors make it unlikely that a practical quantum advantage can be found in this instance.

5.4 Nonconvex optimization: Escaping saddle points and finding local minima

Overview

Finding the global minimum of nonconvex optimization problems is challenging because local algorithms get stuck in local minima. In analogy to physics where the objective function is the energy of the system, these local minima are stable configurations that locally optimize the energy but do not achieve the globally minimal energy. Often, there are many local minima and they are each separated by large energy barriers. Accordingly, instead of finding the global minimum, one may settle for finding a local minimum: local minima can often still be used effectively in situations such as training machine learning models. An effective approach to finding a local minimum is gradient descent, but gradient descent can run into the problem of getting stuck near saddle points, which are not local minima but nonetheless have a vanishing gradient. Efficiently finding local minima thus requires methods for escaping saddle points. Limited work in this area suggests a potential polynomial quantum speedup [1081] in the dimension dependence for finding local minima, using subroutines for Hamiltonian simulation and quantum gradient estimation.

Actual end-to-end problem(s) solved

Suppose we have a classical algorithm $\mathcal{A}_f$ for (approximately) computing a function $f : \mathbb{R}^n \to \mathbb{R}$ which requires C_f gates to perform with a reversible classical circuit. The amount of error tolerable is specified later. Following [1081], suppose further that f is ℓ-smooth and ρ-Hessian Lipschitz, that is,

$$\begin{aligned} \|\nabla f(x_1) - \nabla f(x_2)\| &\leq \ell \|x_1 - x_2\| \qquad \forall x_1, x_2 \in \mathbb{R}^n \\ \|\nabla^2 f(x_1) - \nabla^2 f(x_2)\| &\leq \rho \|x_1 - x_2\| \qquad \forall x_1, x_2 \in \mathbb{R}^n , \end{aligned}$$

where ∇f denotes the gradient of f (a vector), $\nabla^2 f$ denotes the Hessian of f (a matrix), and the norm notation $\|\cdot\|$ denotes the standard Euclidean norm for vector arguments and the spectral norm for matrix arguments.

The end-to-end problem solved is to take as input a specification of the function f, an initial point x_0, and an error parameter ϵ, and to output an ϵ-approximate second-order stationary point (i.e., approximate local minimum) x, defined as satisfying

$$\|\nabla f(x)\| \leq \epsilon \qquad\qquad \lambda_{\min}(\nabla^2 f(x)) \geq -\sqrt{\rho\epsilon} ,$$

where $\lambda_{\min}(\cdot)$ denotes the minimum eigenvalue of its argument. In other words, the gradient should be nearly zero, and the Hessian should be close to a positive-semidefinite matrix.

Dominant resource cost/complexity

The idea pursued in the quantum algorithm of [1081] is to run normal gradient descent, which has gradient query cost independent of n, until reaching an approximate saddle point. Classical algorithms typically apply random perturbations to detect a direction of negative curvature and continue the gradient descent. Instead, the quantum algorithm constructs a Gaussian wavepacket localized at the saddle point, and evolves according to the Schrödinger equation

$$\mathrm{i}\frac{\partial}{\partial t}\Phi = \left(-\frac{1}{2}\Delta + f(x)\right)\Phi\,, \tag{5.1}$$

where Δ denotes the Laplacian operator. The intuition is that, in the directions of positive curvature, the particle stays localized (as in a harmonic potential), while in the directions of negative curvature, the particle quickly disperses. Thus, when the position of the particle is measured, it is likely to have escaped the saddle point in a direction of negative curvature, and gradient descent can be continued. The other technical ingredient is the quantum gradient estimation algorithm, which uses a constant number of (coherent) queries to the function f to estimate ∇f.

The main finding of [1081] is that, to do the gradient descent and the Hamiltonian simulation, the algorithm need only query the circuit evaluating the function f polylog(n) times. Specifically, the algorithm performs the C_f-gate quantum circuit for coherently computing f a number of times scaling as

$$\widetilde{\mathcal{O}}\left(\frac{\log(n)(f(x_0) - f^*)}{\epsilon^{1.75}}\right),$$

where x_0 is the initial point and f^* is the global minimum of f. The evaluation of f must be correct up to precision $\mathcal{O}(\epsilon^2/n^4)$. Note that the work of [1081] initially showed a $\log^2(n)$ dependence, which was later improved to $\log(n)$ using the improved simulation method of [287, Corollary 8]. However, it is important to emphasize that the method has additional cost beyond the queries to the circuit for evaluating f, originating from the Hamiltonian simulation of the kinetic term $-\frac{1}{2}\Delta$ in Eq. (5.1). Specifically, the number of additional gates needed in the Hamiltonian simulation is seen from [287, Lemma 12 & Corollary 8] to scale as $\widetilde{\mathcal{O}}(n(f(x_0) - f^*)/\epsilon^{1.75})$, although we remark that it is possible that this gate complexity could be parallelized such that the circuit depth scales polylogarithmically in n. Thus, the overall gate complexity is

$$\widetilde{\mathcal{O}}\left(\frac{(n + C_f \log(n))(f(x_0) - f^*)}{\epsilon^{1.75}}\right).$$

The space complexity to represent the d-dimensional system on a grid with grid spacing $\mathcal{O}(\epsilon)$ is $\mathcal{O}(n\log(1/\epsilon))$.

In related work, [439] analyzes the complexity of escaping a saddle point when one has access to *noisy* queries to the value of the function f. Additionally, lower bounds on the ϵ-dependence of quantum algorithms for this problem are given in [1080].

Existing resource estimates

This problem has received relatively little attention, and no resource estimates have been performed.

Caveats

Reference [1081] gives the query complexity of the quantum algorithm but does not perform a full end-to-end resource analysis. (However, it does numerically study the performance of the quantum algorithm in a couple of toy examples.) Additionally, many practical scenarios are said to obey a "cheap gradient principle" [457, 163], which says that computing the gradient is almost as easy as computing the function itself, and in these scenarios, no significant quantum speedup is available. Finally, in the setting of variational quantum algorithms, this does not avoid the issue of barren plateaus, which refers to the situation where a large portion of the parameter space has a gradient (and Hessian) that vanishes exponentially with n. These regions would be characterized as ϵ-approximate local minima unless ϵ is made exponentially small in n.

Comparable classical complexity and challenging instance sizes

The best classical algorithm [1079] for this problem makes

$$\widetilde{O}\left(\frac{\log(n)(f(x_0) - f^*)}{\epsilon^{1.75}}\right)$$

queries to the *gradient* of f. Note that $\Omega(n)$ queries to the value of f would be needed to construct a query to the gradient. (When the quantum algorithm in [1081] was first discovered, the best classical algorithm required $O(\log(n)^6)$ gradient queries [577, Theorem 3], and this was later improved.) Although the literature focuses mainly on the query complexity, examination of [1079, Algorithm 1] indicates that an additional $\widetilde{O}(n(f(x_0)-f^*)/\epsilon^{1.75})$ arithmetic operations would be required to process the results of the gradient queries and compute the next point to be queried (e.g., adding pairs of n-dimensional vectors).

Speedup

The quantum algorithm in [1081] has the same query complexity as the classical algorithm in [1079]; the difference is that the quantum algorithm makes (coherent) queries to an evaluation oracle, while the classical algorithm requires access to a gradient oracle. Thus, if classical gradient queries are just as cheap as evaluation queries (as is often the case), there is no speedup. If it were the case that gradient queries are not directly available, then the speedup in query complexity could be exponential. However, even in this case, the speedup in gate complexity can be at most polynomial, since both the classical and quantum algorithms have poly(n) gate complexity, and classical gradient queries can be constructed from $\widetilde{O}(n)$ classical evaluation queries, which can be accomplished with at most C_f classical gates. Indeed, the largest polynomial speedup in gate complexity occurs when $C_f = O(n)$—in this case, the quantum algorithm has $\widetilde{O}(n)$ gate complexity and the classical algorithm has $\widetilde{O}(n) \cdot C_f = O(n^2)$ gate complexity, a quadratic speedup.

Outlook

It is unclear whether the algorithm for finding local minima could lead to a practical speedup, as it depends highly on the (non)availability of an efficient classical procedure for implementing gradient oracles; a quantum speedup is possible only when such oracles are difficult to implement classically, and even so, the speedup in gate complexity would be modest. However, the algorithm represents a useful end-to-end problem where the quantum gradient estimation primitive can be applied. It is also notable that the quantum algorithm employs Hamiltonian simulation, a primitive not used in most other approaches to continuous optimization. Relatedly, [675, 674] propose a quantum subroutine called "quantum Hamiltonian descent" which is a genuinely quantum counterpart to classical gradient descent, via Hamiltonian simulation of an equation similar to Eq. (5.1). Unlike classical gradient descent, it can exploit quantum tunneling to avoid getting stuck in local minima; thus, it can potentially find *global* minima of nonconvex functions. Establishing concrete end-to-end problems where quantum approaches based on Hamiltonian simulation yield an advantage in nonconvex optimization is an interesting direction for future work.