Optimisation of Quantum Hamiltonian Evolution

Apoorva Patel

Centre for High Energy Physics and

Supercomputer Education and Research Centre

Indian Institute of Science, Bangalore

26 June 2014, Lattice 2014, New York

Motivation

Richard Feynman: Quantum computers are efficient simulators of quantum physical systems and models.

Classical simulations of quantum systems and models are not efficient.

Quantum superposition can sum multiple evolutionary paths contributing to a quantum process in one go, while classical simulators evaluate them one by one.

Motivation

Richard Feynman: Quantum computers are efficient simulators of quantum physical systems and models.

Classical simulations of quantum systems and models are not efficient.

Quantum superposition can sum multiple evolutionary paths contributing to a quantum process in one go, while classical simulators evaluate them one by one.

This advantage needs to be formalised in terms of computational complexity, for physical Hamiltonians.

Feynman (1982), Lloyd (1996), Aharanov and Ta-Shma (2003), Berry et al. (2007, 2013)

Motivation

Richard Feynman: Quantum computers are efficient simulators of quantum physical systems and models.

Classical simulations of quantum systems and models are not efficient.

Quantum superposition can sum multiple evolutionary paths contributing to a quantum process in one go, while classical simulators evaluate them one by one.

This advantage needs to be formalised in terms of computational complexity, for physical Hamiltonians.

Feynman (1982), Lloyd (1996), Aharanov and Ta-Shma (2003), Berry et al. (2007, 2013)

Computational complexity of a problem is a measure of the physical resources required to solve it.

Space Time Energy

Tradeoffs between resources are dictated by their availability, e.g. parallel computers.

Conventional classification does not explicitly include energy.

Computational Complexity

Computational complexity of a decision problem is specified in terms of the size of its input (output size is only one bit).

Problems with different output structures are reformulated as a sequence of decision problems, with successive verifiable bounds on the outputs (e.g. as in binary search).

For a specified tolerance level ϵ, the corresponding output size is $\log \epsilon$.

The complexity of the original problem is then the sum of complexities of the individual decision problems.

Computational Complexity

Computational complexity of a decision problem is specified in terms of the size of its input (output size is only one bit).

Problems with different output structures are reformulated as a sequence of decision problems, with successive verifiable bounds on the outputs (e.g. as in binary search).

For a specified tolerance level ϵ, the corresponding output size is $\log \epsilon$.

The complexity of the original problem is then the sum of complexities of the individual decision problems.

It is therefore appropriate to specify the complexity of a general problem in terms of its input and output sizes.

This is a natural criterion for reversible computation. It is also suitable for extending finite precision analog computation to arbitrary precision digital computation.

A computational algorithm is efficient when the required resources are polynomial in its input and output sizes.

Popular importance sampling methods are not efficient, with error $\epsilon \propto N_{\rm iter}^{-1/2}$.

Quantum Hamiltonian Simulation

Start from the initial quantum state $|\psi(0)\rangle$.

First evolve: $|\psi(T)\rangle = U(T)|\psi(0)\rangle, \;\; U(T) = P[e^{-i\int_0^T H(t)dt}]$.
Then measure: $\langle O_a\rangle = \langle\psi(T)|O_a|\psi(T)\rangle$.

In typical problems, both these parts are executed probabilistically upto a specified tolerance level, say ϵ.

We address the first part: The problem is to determine the evolution operator $U(T)$, with accuracy $||\widetilde{U}(T) - U(T)|| < \epsilon$.

Efficient execution of the second part requires different techniques.

Quantum Hamiltonian Simulation

Start from the initial quantum state $|\psi(0)\rangle$.

First evolve: $|\psi(T)\rangle = U(T)|\psi(0)\rangle, \;\; U(T) = P[e^{-i\int_0^T H(t)dt}]$.
Then measure: $\langle O_a\rangle = \langle\psi(T)|O_a|\psi(T)\rangle$.

In typical problems, both these parts are executed probabilistically upto a specified tolerance level, say ϵ.

We address the first part: The problem is to determine the evolution operator $U(T)$, with accuracy $||\widetilde{U}(T) - U(T)|| < \epsilon$.

Efficient execution of the second part requires different techniques.

In a finite N-dimensional Hilbert space, a generic $H(t)$ is a dense $N \times N$ matrix. That cannot be simulated efficiently.

Physical properties restrict the structure of $H(t)$, however. Efficient simulations must exploit this Hamiltonian structure.

Useful Physical Features

Features commonly present in physical problems are:

(1) The Hilbert space is a tensor product of many small components (e.g. $N = 2^n$ for a system of qubits).

Useful Physical Features

Features commonly present in physical problems are:

(1) The Hilbert space is a tensor product of many small components (e.g. $N = 2^n$ for a system of qubits).

(2) The components have only local interactions (e.g. couplings with only a limited number of neighbours).

Such sparse Hamiltonians have $O(N)$ non-zero elements.

Exception: An easily factorisable dense operator is also fine, as in case of FFT.

Useful Physical Features

Features commonly present in physical problems are:

(1) The Hilbert space is a tensor product of many small components (e.g. $N = 2^n$ for a system of qubits).

(2) The components have only local interactions (e.g. couplings with only a limited number of neighbours).

Such sparse Hamiltonians have $O(N)$ non-zero elements.

Exception: An easily factorisable dense operator is also fine, as in case of FFT.

(3) The Hamiltonian is specified using a finite number of functions (e.g. translationally invariant interactions).

With such a compact description of the Hamiltonian, the resources needed to just write it do not influence the simulation complexity.

Useful Physical Features

Features commonly present in physical problems are:

(1) The Hilbert space is a tensor product of many small components (e.g. $N = 2^n$ for a system of qubits).

(2) The components have only local interactions (e.g. couplings with only a limited number of neighbours).

Such sparse Hamiltonians have $O(N)$ non-zero elements.

Exception: An easily factorisable dense operator is also fine, as in case of FFT.

(3) The Hamiltonian is specified using a finite number of functions (e.g. translationally invariant interactions).

With such a compact description of the Hamiltonian, the resources needed to just write it do not influence the simulation complexity.

Such Hamiltonians can be mapped to graphs with bounded degree d (vertices $\leftrightarrow$ components, edges $\leftrightarrow$ interactions).

Above features permit SIMD simulations of these Hamiltonians with domain decomposition.

Useful Physical Features

Features commonly present in physical problems are:

(1) The Hilbert space is a tensor product of many small components (e.g. $N = 2^n$ for a system of qubits).

(2) The components have only local interactions (e.g. couplings with only a limited number of neighbours).

Such sparse Hamiltonians have $O(N)$ non-zero elements.

Exception: An easily factorisable dense operator is also fine, as in case of FFT.

(3) The Hamiltonian is specified using a finite number of functions (e.g. translationally invariant interactions).

With such a compact description of the Hamiltonian, the resources needed to just write it do not influence the simulation complexity.

Such Hamiltonians can be mapped to graphs with bounded degree d (vertices $\leftrightarrow$ components, edges $\leftrightarrow$ interactions).

Above features permit SIMD simulations of these Hamiltonians with domain decomposition.

Efficient Hamiltonian simulation algorithms use resources that are polynomial in $\log(N)$, d and $\log(\epsilon)$.

Evolution Strategy

Efficient simulation strategy has two major ingredients:

(A) Decompose the sparse Hamiltonian as a sum of non-commuting but block-diagonal parts, $H = \sum_{i=1}^{l} H_i$.

Then each H_i can be easily and exactly exponentiated, with $\exp(-iH_i\tau)$ retaining the block-diagonal structure.

The smallest possible blocks are of size 2×2: $H^{(b)} = a_0 I + \vec{a} \cdot \vec{\sigma}$.

Their projection operator structure allows them to be interpreted as binary query oracles.

Evolution Strategy

Efficient simulation strategy has two major ingredients:

(A) Decompose the sparse Hamiltonian as a sum of non-commuting but block-diagonal parts, $H = \sum_{i=1}^{l} H_i$.

Then each H_i can be easily and exactly exponentiated, with $\exp(-iH_i\tau)$ retaining the block-diagonal structure.

The smallest possible blocks are of size 2×2: $H^{(b)} = a_0 I + \vec{a} \cdot \vec{\sigma}$.

Their projection operator structure allows them to be interpreted as binary query oracles.

H_i can be identified by an edge-colouring algorithm for graphs, with distinct colours for overlapping edges.
Any graph can be efficiently coloured with $d + 1$ colours.

Algorithms for bipartite graphs are simpler than the generic case. They need d colours.

Evolution Strategy

Efficient simulation strategy has two major ingredients:

(A) Decompose the sparse Hamiltonian as a sum of non-commuting but block-diagonal parts, $H = \sum_{i=1}^{l} H_i$.

Then each H_i can be easily and exactly exponentiated, with $\exp(-iH_i\tau)$ retaining the block-diagonal structure.

The smallest possible blocks are of size 2×2: $H^{(b)} = a_0 I + \vec{a} \cdot \vec{\sigma}$.

Their projection operator structure allows them to be interpreted as binary query oracles.

H_i can be identified by an edge-colouring algorithm for graphs, with distinct colours for overlapping edges.
Any graph can be efficiently coloured with $d + 1$ colours.

Algorithms for bipartite graphs are simpler than the generic case. They need d colours.

Identification of H_i provides a compressed labeling scheme to address individual blocks.
The blocks can then be easily evolved in parallel (classically), or in superposition (quantum mechanically).

Example of Hamiltonian decomposition:

Discretised Laplacian in 1-dim can be decomposed as:

$$\begin{pmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & -1 & 2 & -1 & 0 & 0 & \cdots \\ \cdots & 0 & -1 & 2 & -1 & 0 & \cdots \\ \cdots & 0 & 0 & -1 & 2 & -1 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix} =$$

$$\begin{pmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & -1 & 1 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 1 & -1 & 0 & \cdots \\ \cdots & 0 & 0 & -1 & 1 & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix} + \begin{pmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & 0 & 1 & -1 & 0 & 0 & \cdots \\ \cdots & 0 & -1 & 1 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 1 & -1 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix}$$

This decomposition has the projection operator structure following from: $H = H_o + H_e,\ H_o^2 = 2H_o,\ H_e^2 = 2H_e$.

Example of Hamiltonian decomposition:

Discretised Laplacian in 1-dim can be decomposed as:

$$\begin{pmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & -1 & 2 & -1 & 0 & 0 & \cdots \\ \cdots & 0 & -1 & 2 & -1 & 0 & \cdots \\ \cdots & 0 & 0 & -1 & 2 & -1 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix} =$$

$$\begin{pmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & -1 & 1 & 0 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 1 & -1 & 0 & \cdots \\ \cdots & 0 & 0 & -1 & 1 & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix} + \begin{pmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & 0 & 1 & -1 & 0 & 0 & \cdots \\ \cdots & 0 & -1 & 1 & 0 & 0 & \cdots \\ \cdots & 0 & 0 & 0 & 1 & -1 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{pmatrix}$$

This decomposition has the projection operator structure following from: $H = H_o + H_e,\ H_o^2 = 2H_o,\ H_e^2 = 2H_e$.

Graphically, the bipartite break-up is:

H_o and H_e are identified by the last bit of the position label.

Eigenvalues of H are $4\sin^2(k/2)$. Those of H_o, H_e are $0, 2$.

Evolution Strategy (contd.)

(B) Use the discrete Lie-Trotter formula to exponentiate H, but with as large Δt as possible.

$$e^{-iHT} = e^{-i\sum_{i=1}^{l} H_i T} \approx (\prod_i e^{-iH_i \Delta t})^n, \; n = T/\Delta t$$

This replacement maintains unitarity of the evolution, but may not preserve other properties such as the energy.

Time-dependent Hamiltonians should be expanded about the mid-point of the interval Δt.

Evolution Strategy (contd.)

(B) Use the discrete Lie-Trotter formula to exponentiate H, but with as large Δt as possible.

$$e^{-iHT} = e^{-i\sum_{i=1}^{l} H_i T} \approx (\prod_i e^{-iH_i \Delta t})^n, \; n = T/\Delta t$$

This replacement maintains unitarity of the evolution, but may not preserve other properties such as the energy.

Time-dependent Hamiltonians should be expanded about the mid-point of the interval Δt.

When the exponent is proportional to a projection operator, the largest Δt makes the exponential a reflection operator.

$$P = \tfrac{1}{2}(1 - \hat{n}\cdot\vec{\sigma}), \; P^2 = P \;\Rightarrow\; R = e^{\pm i\pi P} = 1 - 2P = \hat{n}\cdot\vec{\sigma}, \; R^2 = I.$$

Evolution Strategy (contd.)

(B) Use the discrete Lie-Trotter formula to exponentiate H, but with as large Δt as possible.

$$e^{-iHT} = e^{-i\sum_{i=1}^{l} H_i T} \approx (\prod_i e^{-iH_i \Delta t})^n, \; n = T/\Delta t$$

This replacement maintains unitarity of the evolution, but may not preserve other properties such as the energy.

Time-dependent Hamiltonians should be expanded about the mid-point of the interval Δt.

When the exponent is proportional to a projection operator, the largest Δt makes the exponential a reflection operator.

$$P = \tfrac{1}{2}(1 - \hat{n}\cdot\vec{\sigma}), \; P^2 = P \;\Rightarrow\; R = e^{\pm i\pi P} = 1 - 2P = \hat{n}\cdot\vec{\sigma}, \; R^2 = I.$$

This extreme strategy not only keeps the evolution accurate, but also improves the algorithmic complexity from a power law dependence on ϵ to a logarithmic one.

That is not at all obvious, and needs to be demonstrated.

Illustration: Database Search

View database search as a Hamiltonian evolution problem.

The evolution is from the initial uniform superposition state $|s\rangle$ to a specific target state $|t\rangle$: $U(T)|s\rangle = |t\rangle$.

For a database of size N: $|\langle i|s\rangle| = 1/\sqrt{N}$, $\langle i|t\rangle = \delta_{it}$.

Illustration: Database Search

View database search as a Hamiltonian evolution problem.

The evolution is from the initial uniform superposition state $|s\rangle$ to a specific target state $|t\rangle$: $U(T)|s\rangle = |t\rangle$.

For a database of size N: $|\langle i|s\rangle| = 1/\sqrt{N},\ \langle i|t\rangle = \delta_{it}$.

Logic: Design a Hamiltonian to diffuse the wavefunction over the whole Hilbert space by kinetic energy (mean field version) and attract it towards the target by potential energy.

Illustration: Database Search

View database search as a Hamiltonian evolution problem.

The evolution is from the initial uniform superposition state $|s\rangle$ to a specific target state $|t\rangle$: $U(T)|s\rangle = |t\rangle$.

For a database of size N: $|\langle i|s\rangle| = 1/\sqrt{N},\ \langle i|t\rangle = \delta_{it}$.

Logic: Design a Hamiltonian to diffuse the wavefunction over the whole Hilbert space by kinetic energy (mean field version) and attract it towards the target by potential energy.

In simplest algorithms, the Hamiltonians depend only on $|s\rangle$ and $|t\rangle$. The unitary evolution is then a rotation in the 2-dim subspace formed by $|s\rangle$ and $|t\rangle$. Let

$$|t\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix},\ |t_\perp\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix},\ |s\rangle = \begin{pmatrix} 1/\sqrt{N} \\ \sqrt{(N-1)/N} \end{pmatrix}.$$

A time-independent H rotates the state at a fixed rate:

$$|\psi\rangle \to U(t)|\psi\rangle,\ U(t) = \exp(-iHt) = \exp(-i\hat{n}_H \cdot \vec{\sigma}\omega t).$$

Farhi-Gutmann version:

Continuous time evolution with $H_C = |s\rangle\langle s| + |t\rangle\langle t|$ gives:

$U(t) = \exp(-i\hat{n}\cdot\vec{\sigma}t/\sqrt{N}),\ \hat{n} = (\sqrt{(N-1)/N}, 0, 1/\sqrt{N})^T.$

The (unnormalised) eigenvectors of H are $|s\rangle \pm |t\rangle$.
The rotation axis $\hat{n}$ bisects the initial and target states.

Rotation by angle π on the Bloch sphere takes $|s\rangle$ to $|t\rangle$, with evolution time $T = (\pi/2)\sqrt{N}$.

Farhi-Gutmann version:

Continuous time evolution with $H_C = |s\rangle\langle s| + |t\rangle\langle t|$ gives:

$U(t) = \exp(-i\hat{n}\cdot\vec{\sigma}t/\sqrt{N}),\ \hat{n} = (\sqrt{(N-1)/N}, 0, 1/\sqrt{N})^T.$

The (unnormalised) eigenvectors of H are $|s\rangle \pm |t\rangle$.
The rotation axis $\hat{n}$ bisects the initial and target states.

Rotation by angle π on the Bloch sphere takes $|s\rangle$ to $|t\rangle$, with evolution time $T = (\pi/2)\sqrt{N}$.

Grover version:

Time evolution is discrete with the evolution operator

$U_G = -(I - 2|s\rangle\langle s|)(I - 2|t\rangle\langle t|) = (1 - \frac{2}{N})I + 2i\frac{\sqrt{N-1}}{N}\sigma_2\ .$

$U_G = \exp(-iH_G\tau)$ corresponds to the Hamiltonian

$H_G = i([|t\rangle\langle t|, |s\rangle\langle s|] = i(|t\rangle\langle s| - |s\rangle\langle t|)/\sqrt{N} = -\frac{\sqrt{N-1}}{N}\sigma_2\ .$

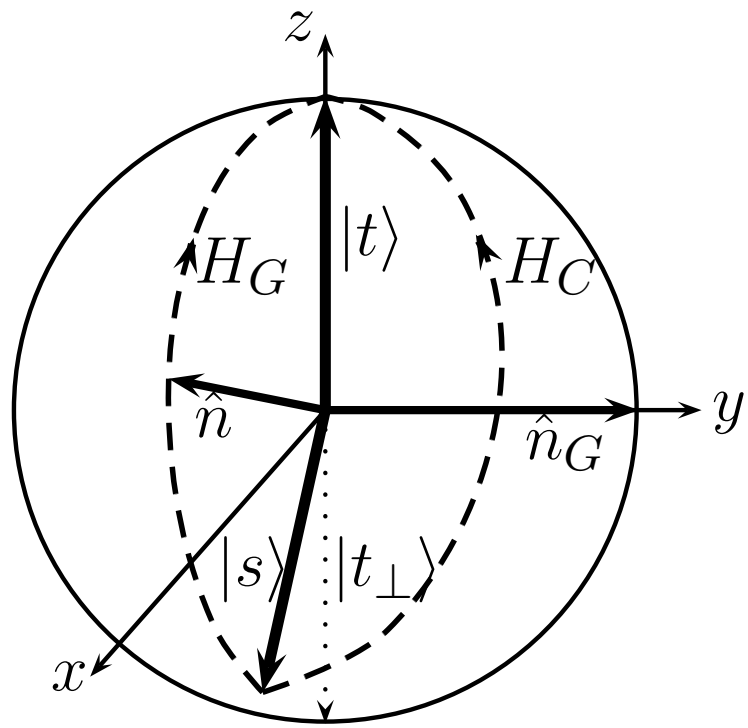
It is the discrete Lie-Trotter formula for H_s and H_t with $\Delta t_G = \pi$. The rotation axis $\hat{n}_G = (0, 1, 0)^T$ is orthogonal to $\hat{n}$.

The evolution time step is: $\tau = \frac{2N}{\sqrt{N-1}} \sin^{-1} \frac{1}{\sqrt{N}}$.

Going from $|s\rangle$ to $|t\rangle$ requires Q steps along the geodesic:
$(U_G)^Q|s\rangle = |t\rangle,\ Q_T = \frac{1}{4}\cos^{-1}(\frac{2}{N} - 1)/\sin^{-1}(1/\sqrt{N}) \approx \frac{\pi}{4}\sqrt{N}.$

The evolution time step is: $\tau = \frac{2N}{\sqrt{N-1}} \sin^{-1} \frac{1}{\sqrt{N}}$.

Going from $|s\rangle$ to $|t\rangle$ requires Q steps along the geodesic:
$(U_G)^Q|s\rangle = |t\rangle,\ Q_T = \frac{1}{4}\cos^{-1}(\frac{2}{N} - 1)/\sin^{-1}(1/\sqrt{N}) \approx \frac{\pi}{4}\sqrt{N}.$



The two evolution trajectories are completely different. Only after a specific evolution time, corresponding to the solution of the search problem, they meet each other.

Adiabatic evolution follows the same trajectory as H_G.

Equivalent Evolutions

For database search: $U_C(T) = i(1 - 2|t\rangle\langle t|)(U_G)^{Q_T}$

For a more general evolution time $0 < t < T$, we have (analogous to the Euler angle decomposition):

$$U_C(t) = \exp(i\beta\sigma_3)\ (U_G)^{Q_t}\ \exp\left(i(\tfrac{\pi}{2} + \beta)\sigma_3\right),$$

$$Q_t = \frac{\sin^{-1}\left(\sqrt{\frac{N-1}{N}}\sin(t/\sqrt{N})\right)}{2\sin^{-1}(1/\sqrt{N})} \approx \frac{t}{2}\ ,\ \sigma_3 = 2|t\rangle\langle t| - 1,$$

$$\beta = -\frac{\pi}{4} - \frac{1}{2}\tan^{-1}\left(\frac{1}{\sqrt{N}}\tan(t/\sqrt{N})\right).$$

Equivalent Evolutions

For database search: $U_C(T) = i(1 - 2|t\rangle\langle t|)(U_G)^{Q_T}$

For a more general evolution time $0 < t < T$, we have (analogous to the Euler angle decomposition):

$$U_C(t) = \exp(i\beta\sigma_3)\ (U_G)^{Q_t}\ \exp\left(i(\tfrac{\pi}{2} + \beta)\sigma_3\right),$$

$$Q_t = \frac{\sin^{-1}\left(\sqrt{\frac{N-1}{N}}\sin(t/\sqrt{N})\right)}{2\sin^{-1}(1/\sqrt{N})} \approx \frac{t}{2}\ ,\ \sigma_3 = 2|t\rangle\langle t| - 1,$$

$$\beta = -\frac{\pi}{4} - \frac{1}{2}\tan^{-1}\left(\frac{1}{\sqrt{N}}\tan(t/\sqrt{N})\right).$$

Thus $U_C(t)$ can be expressed entirely in terms of projection operators, and the two evolutions are identical irrespective of the initial state and the evolution time.

H_G can be used to obtain the same evolution as H_C, even though they have different eigenvectors and eigenvalues.

Fractional oracle operator, $O_\phi = exp(i\phi|t\rangle\langle t|)$, is easily generated using an ancilla bit.

Complexity of Discretised Evolution

All continuous variables are discretised in digital computers.

That is needed for implementing fault-tolerant computation with control over bounded errors.

Discrete evolution step Δt has to be chosen so as to satisfy the overall error bound ϵ on the algorithm.

Complexity of Discretised Evolution

All continuous variables are discretised in digital computers.

That is needed for implementing fault-tolerant computation with control over bounded errors.

Discrete evolution step Δt has to be chosen so as to satisfy the overall error bound ϵ on the algorithm.

The simplest and the symmetric Lie-Trotter formulae are:

$$
\begin{aligned}
e^{-i\sum_{i=1}^{l} H_i \Delta t} &= (e^{-iH_1\Delta t} ... e^{-iH_l\Delta t}) \times e^{-iE^{(2)}(\Delta t)^2} \\
e^{-i\sum_{i=1}^{l} H_i \Delta t} &= (e^{-iH_l\Delta t/2} ... e^{-iH_1\Delta t/2}) \\
&\times (e^{-iH_1\Delta t/2} ... e^{-iH_l\Delta t/2}) \times e^{-iE^{(3)}(\Delta t)^3}
\end{aligned}
$$

with discretisation errors:

$$
\begin{aligned}
E^{(2)} &= \frac{i}{24}\sum_{i<j}[H_i, H_j] + O(\Delta t) \\
E^{(3)} &= \frac{1}{24}\sum_{i<j}\{2[H_i,[H_i,H_j]] + [H_j,[H_i,H_j]]\} \\
&+ \frac{1}{12}\sum_{i<j<k}\{2[H_i,[H_j,H_k]] + [H_j,[H_i,H_k]]\} + O(\Delta t)
\end{aligned}
$$

Small step size Δt:

For unitary operators, $||X|| = 1$. For n evolution steps, triangle and Cauchy-Schwarz inequalities bound the error:
$||X^n - Y^n|| = ||(X - Y)(X^{n-1} + \ldots + Y^{n-1})|| \leq n||X - Y||$.

So to keep the total discretisation error bounded, we need
$n||e^{-iE^{(k)}(\Delta t)^k} - I|| \approx n(\Delta t)^k||E^{(k)}|| = t(\Delta t)^{k-1}||E^{(k)}|| < \epsilon$.

With exact exponentiation of the individual H_i, the computational cost $\mathcal{C}$ of a single step is independent of Δt.

Small step size Δt:

For unitary operators, $||X|| = 1$. For n evolution steps, triangle and Cauchy-Schwarz inequalities bound the error:
$||X^n - Y^n|| = ||(X - Y)(X^{n-1} + \ldots + Y^{n-1})|| \leq n||X - Y||$.

So to keep the total discretisation error bounded, we need
$n||e^{-iE^{(k)}(\Delta t)^k} - I|| \approx n(\Delta t)^k||E^{(k)}|| = t(\Delta t)^{k-1}||E^{(k)}|| < \epsilon$.

With exact exponentiation of the individual H_i, the computational cost $\mathcal{C}$ of a single step is independent of Δt.

The computational complexity of the whole evolution is then

$$O(n\mathcal{C}) = O(t^{k/(k-1)}(||E^{(k)}||/\epsilon)^{1/(k-1)}\mathcal{C}).$$

With power-law scaling in ϵ, this scheme is inefficient.

For the Hamiltonian H_C, $||E^{(2)}||$ and $||E^{(3)}||$ are $O(N^{-1/2})$.
For evolution time $T = \Theta(N^{1/2})$, its time complexity is linear.

Grover's discretisation:

Δt_G is chosen to make $\exp(-H_i \Delta t_G)$ reflection operators.

The large step size introduces an error because one may jump across the desired state instead of reaching it exactly. In general, Q_t is not an integer, and has to be replaced by its nearest integer approximation $\lfloor Q_t + \frac{1}{2} \rfloor$.

Grover's discretisation:

Δt_G is chosen to make $\exp(-H_i \Delta t_G)$ reflection operators.

The large step size introduces an error because one may jump across the desired state instead of reaching it exactly. In general, Q_t is not an integer, and has to be replaced by its nearest integer approximation $\lfloor Q_t + \frac{1}{2} \rfloor$.

The error probability for $U_C(t)$ is thus bounded by $1/N$, corresponding to half a rotation step. Simple repetition of the algorithm and selection of the result by majority rule (not average) can rapidly reduce the error probability.

With R repetitions, the error probability is less than $(N/4)^R$, which can be made smaller than any prescribed ϵ.

Grover's discretisation:

Δt_G is chosen to make $\exp(-H_i \Delta t_G)$ reflection operators.

The large step size introduces an error because one may jump across the desired state instead of reaching it exactly. In general, Q_t is not an integer, and has to be replaced by its nearest integer approximation $\lfloor Q_t + \frac{1}{2} \rfloor$.

The error probability for $U_C(t)$ is thus bounded by $1/N$, corresponding to half a rotation step. Simple repetition of the algorithm and selection of the result by majority rule (not average) can rapidly reduce the error probability.

With R repetitions, the error probability is less than $(N/4)^R$, which can be made smaller than any prescribed ϵ.

The computational complexity of the total evolution is then

$$O(Q_t R \mathcal{C}_G) = O(\tfrac{t}{2}(-\tfrac{2\log\epsilon}{\log N})\mathcal{C}_G) = O(-t\tfrac{\log\epsilon}{\log N}\mathcal{C}_G),$$

and the algorithm is efficient.

Key Features

With a straightforward application of the Lie-Trotter formula, the algorithm has an error proportional to the number of steps n, and a power-law dependence of complexity on ϵ.

With the Lie-Trotter formula based on exact exponentiation of projection operators to reflection operators, the algorithm has an error independent of the evolution time, and a logarithmic dependence of complexity on ϵ.

Key Features

With a straightforward application of the Lie-Trotter formula, the algorithm has an error proportional to the number of steps n, and a power-law dependence of complexity on ϵ.

With the Lie-Trotter formula based on exact exponentiation of projection operators to reflection operators, the algorithm has an error independent of the evolution time, and a logarithmic dependence of complexity on ϵ.

Algebraically, the Baker-Campbell-Hausdorff expansion reduces to a finite number of terms for projection operators. That allows efficient implementation of the Lie-Trotter formula even for large step size.

Key Features

With a straightforward application of the Lie-Trotter formula, the algorithm has an error proportional to the number of steps n, and a power-law dependence of complexity on ϵ.

With the Lie-Trotter formula based on exact exponentiation of projection operators to reflection operators, the algorithm has an error independent of the evolution time, and a logarithmic dependence of complexity on ϵ.

Algebraically, the Baker-Campbell-Hausdorff expansion reduces to a finite number of terms for projection operators. That allows efficient implementation of the Lie-Trotter formula even for large step size.

With compressed labeling, operations on specific blocks are easily implemented as controlled unitary operations.

Euler angle decomposition allows easy conversion of rotations about arbitrary axes to rotations about fixed axes.

Truncation Error

A digital computer with finite number of bits produces truncation errors. With b bits, the precision is $\delta = 2^{-b}$.

Addition, multiplication and polynomial evaluations respectively require $O(b)$, $O(b^2)$ and $O(b^3)$ resources.

Overflow/underflow limits the degree of the polynomial to be at most b.

With all functions approximated by accurate polynomials, fixed axes rotations to b-bit precision need $O(b^3)$ effort.

Truncation Error

A digital computer with finite number of bits produces truncation errors. With b bits, the precision is $\delta = 2^{-b}$.

Addition, multiplication and polynomial evaluations respectively require $O(b)$, $O(b^2)$ and $O(b^3)$ resources.

Overflow/underflow limits the degree of the polynomial to be at most b.

With all functions approximated by accurate polynomials, fixed axes rotations to b-bit precision need $O(b^3)$ effort.

The number of exponentiations of H_i needed for the Lie-Trotter formula is $n(k-1)l$, which reduces to $2Q_t \approx t$ for the Grover version.

The truncation error can be always made negligible compared to the discretisation error, with the choice $n(k-1)l\delta = O(\epsilon)$, i.e. $b = \Theta(-\log(\epsilon/n))$.
The cost of a single step then scales as $\mathcal{C} = O((-\log\epsilon)^3)$, and the algorithm remains efficient.

Generalisations

(1) Laplacian evolution can (marginally) beat FFT.

The Hamiltonian is a sum of only two projection operators in any dimension.

Generalisations

(1) Laplacian evolution can (marginally) beat FFT.

The Hamiltonian is a sum of only two projection operators in any dimension.

(2) Overrelaxation evolution is more efficient than evolution with small Metropolis steps.

Generalisations

(1) Laplacian evolution can (marginally) beat FFT.

The Hamiltonian is a sum of only two projection operators in any dimension.

(2) Overrelaxation evolution is more efficient than evolution with small Metropolis steps.

(3) The evolution identity in terms of reflection operators remains exact even when magnitudes of H_i are unequal. Only the parameters Q_t and β change.

Generalisations

(1) Laplacian evolution can (marginally) beat FFT.

The Hamiltonian is a sum of only two projection operators in any dimension.

(2) Overrelaxation evolution is more efficient than evolution with small Metropolis steps.

(3) The evolution identity in terms of reflection operators remains exact even when magnitudes of H_i are unequal. Only the parameters Q_t and β change.

(4) For general Hamiltonians, successive H_i can be added to the algorithm one by one (e.g. in an induction procedure). The large step evolution then is not exact, but has $\Theta(1)$ success probability for a suitable evolution duration.
The overall scaling of the algorithm remains efficient:

$$O(lt||H||\log^3(lt||H||/\epsilon)\mathrm{Poly}(\log N)).$$

Generalisations

(1) Laplacian evolution can (marginally) beat FFT.

The Hamiltonian is a sum of only two projection operators in any dimension.

(2) Overrelaxation evolution is more efficient than evolution with small Metropolis steps.

(3) The evolution identity in terms of reflection operators remains exact even when magnitudes of H_i are unequal. Only the parameters Q_t and β change.

(4) For general Hamiltonians, successive H_i can be added to the algorithm one by one (e.g. in an induction procedure). The large step evolution then is not exact, but has $\Theta(1)$ success probability for a suitable evolution duration.
The overall scaling of the algorithm remains efficient:

$$O(lt||H||\log^3(lt||H||/\epsilon)\mathrm{Poly}(\log N)).$$

(5) Block-diagonal structure of H_i can evaluate many other functions easily, e.g. fermion determinants.

References

R.P. Feynman, Simulating Physics with Computers,
Int. J. Theor. Phys. 21 (1982) 467-488

S. Lloyd, Universal Quantum Simulators,
Science 273 (1996) 1073-1078

D. Aharonov and A. Ta-Shma,
Adiabatic Quantum State Generation and Statistical Zero Knowledge,
Proc. 35th Annual ACM Symp. on Theory of Computing, ACM (2003) 20-29

D.W. Berry, G. Ahokas, R. Cleve and B.C. Sanders,
Efficient Quantum Algorithms for Simulating Sparse Hamiltonians,
Comm. Math. Phys. 270 (2007) 359-371

A.M. Childs and R. Kothari
Simulating Sparse Hamiltonians with Star Decompositions,
Proc. TQC2010, Lecture Notes in Comp. Sci. 6519 (2011) 94-103

D.W. Berry, R. Cleve and R.D. Somma,
Exponential Improvement in Precision for Hamiltonian-Evolution Simulation,
Presented at AQIS'13, Chennai (2013), arXiv:1308.5424

D.W. Berry, A.M. Childs, R. Cleve, R. Kothari and R.D. Somma,
Exponential Improvement in Precision for Simulating Sparse Hamiltonians,
arXiv:1312.1414

H. De Raedt, Product Formula Algorithms for Solving the Time-Dependent Schrödinger Equation, Comp. Phys. Rep. 7 (1987) 1-72

J.L. Richardson, Visualizing Quantum Scattering on the CM-2 Supercomputer, Comp. Phys. Comm. 63 (1991) 84-94

L.K. Grover, From Schrödinger's Equation to the Quantum Search Algorithm, Pramana 56 (2001) 333-348

E. Farhi and S. Gutmann, An Analog Analogue of a Digital Quantum Computation, Phys. Rev. A 57 (1998) 2403-2406

L.K. Grover, A Fast Quantum Mechanical Algorithm for Database Search, Proc. 28th Annual ACM Symp. on Theory of Computing, ACM (1996) 212-219

M.A. Nielsen and I.L. Chuang, Quantum Computation and Quantum Information, Cambridge University Press (2000)