Week 6: Quantum Singular Value Transform

Stacey Jeffery

March 5, 2025

6.1 Introduction: A Grand Unification of Quantum Algorithms

You have no doubt noticed that quantum algorithms are very hard to design. Coming up with truly *new* quantum algorithmic ideas is so difficult that it has often been said that there are only two quantum algorithms: Shor's factoring algorithm (period finding) and Grover's search algorithm – and every other algorithm is just a variation of one of these. This is, of course, incorrect, as there is a third quantum algorithm: Hamiltonian simulation, which we will see shortly.

However, not so long ago (in 2018), a PhD student in Amsterdam and his collaborators, developed a framework that could be said to unify the most important ideas underlying many or all quantum algorithms into a single framework: The Quantum Singular Value Transform (QSVT) Framework. It was described, in a very nice survey [MRTC21], as a "Grand Unification of Quantum Algorithms."

6.2 Some Problems we will Consider this Week

We start by defining some problems we will consider in the QSVT framework, as motivating examples. First, some quick linear algebra background.

A *normal* matrix is a square matrix A such that $A^\dagger A = AA^\dagger$. A matrix is normal if and only if it can be expressed as

$$A = \sum_{j\in\mathcal{J}} \lambda_j \Pi_j, \tag{6.1}$$

where $\{\lambda_j\}_{j\in\mathcal{J}}$ are the eigenvalues of A, and Π_j is the orthogonal projector onto the λ_j-eigenspace. For example, unitary matrices ($A^\dagger A = AA^\dagger = I$) are normal, as are Hermitian matrices ($A = A^\dagger$).

A function $f : \mathbb{C} \to \mathbb{C}$ can be extended to normal matrices by

$$f(A) = \sum_{j\in\mathcal{J}} f(\lambda_j)\Pi_j$$

where A is as in (6.1). A function $f : \mathbb{R} \to \mathbb{C}$ can be extended to Hermitian matrices – which are precisely normal matrices with real eigenvalues – in the same way. For example, if A is Hermitian, and $f(x) = e^{ix}$, then

$$f(A) = \sum_{j\in\mathcal{J}} e^{i\lambda_j}\Pi_j.$$

Since the image of f on $\mathbb{R}$ is the unit circle, $f(A)$ is unitary.

Exercise 6.2.1. *Let $f(x) = 1 - x$. Show that for any normal matrix $A \in \mathbb{C}^{N\times N}$, $f(A) = I - A$, where I is the N-by-N identity.*

6.2.1 Hamiltonian Simulation

Hamiltonian simulation is, loosely speaking, the following problem: Given a *Hamiltonian*,[1] H, which is a Hermitian operator, and some time t, apply the unitary $U = e^{itH}$ to some input state. This simulates what happens to a physical system described by H, that starts in the input state, and is left to evolve for t units of time. There are potential applications of such a simulation, to various problems where simulating a quantum mechanical system would be useful. These range from simulating experiments in order to improve our understanding of physical principles, to more practical applications, like developing new materials, medicines, or catalysts for nitrogen fixation (which is important for making fertilizer). This is just the sort of problem quantum computers were made for: simulating a 2^n-dimensional (n-qubit) Hamiltonian seems to require classical resources that scale exponentially in n, whereas an n-qubit quantum system can efficiently simulate itself – if a Hamiltonian on $\mathbb{C}^{2^n}$ describes a system that occurs in nature, then simulating it for time t takes ... n qubits, and t time. So this is a problem where, at least for special cases, we get an exponential speedup.

Many of the potential applications of Hamiltonian simulation are not algorithms themselves, but rather, the idea that having a better understanding of certain physical or chemical systems would likely lead to new scientific and technological breakthroughs. Some of these are well worked out ideas (for example, nitrogen fixation [RWS+17]), but many of them are very tenuous, which unfortunately does not stop popular science news, and technology-enthusiasts who have decided to make a career talking a lot about quantum computing without really understanding it, from treating such applications as being just around the corner. You will find headlines and ted talks claiming quantum computers can solve every futuristic-sounding problem, including fixing climate change [mtl], curing cancer [Kak24], and finding the secret to immortality [Gre20]. I mean, *science* could solve any of these (but could it?), and faster Hamiltonian simulation would mean we can do better science, so... The reality is, there probably will be many applications to being able to simulate physical systems, but we do not yet know what they will be. We will not discuss applications of Hamiltonian simulation in this course, but it is important to understand that there is a lot of hype around quantum computing, some of which is justified, and some of which is not.

Before we precisely define Hamiltonian simulation, we need the following.

Definition 6.2.1 (Sparse Access)**.** *Let $A \in \mathbb{C}^{M\times N}$. We say A is s-sparse if every row and column has at most s non-zero entries. We say we have* sparse-access *to an s-sparse matrix A if we have oracles $(\mathcal{O}_A, \mathcal{O}_r, \mathcal{O}_c)$ – called* sparse access oracles *– acting as:*

$$\begin{aligned}\forall i \in [M], j \in [N],\ \mathcal{O}_A : |i\rangle|j\rangle|0\rangle \mapsto |i\rangle|j\rangle|A_{i,j}\rangle\\ \forall i \in [M], k \in [s]\ \mathcal{O}_r : |i\rangle|k\rangle \mapsto |i\rangle|r_{i,k}\rangle\\ \forall j \in [N], k \in [s],\ \mathcal{O}_c : |j\rangle|k\rangle \mapsto |j\rangle|c_{j,k}\rangle,\end{aligned}$$

where $r_{i,k} \in [N]$ is the index of the k-th non-zero entry of row i of A, and $c_{j,k} \in [M]$ is the index of the k-th non-zero entry of column j of A. If some row (or column) of A has $s' < s$ non-zero entries, we could have $\mathcal{O}_r|i\rangle|k\rangle$ return some error symbol when $k > s'$, but this is not necessary. We simply need that $\{r_{i,k} : k \in [s]\}$ contains *all indices of non-zero entries of row i.*

We are ignoring the issue of precision of the entries of A, and simply assuming we have a register large enough to store a full description of any entry of A.

Problem: HamSim

---

[1] In this section, H will be a Hamiltonian, so we will use $\mathcal{H}$ for a finite-dimensional Hilbert space (i.e. inner product space).

Input: $t \in \mathbb{Z}_{\geq 0}$, sparse access oracles $(\mathcal{O}_H, \mathcal{O}_r, \mathcal{O}_c)$ for a Hermitian matrix $H \in \mathbb{C}^{2^n \times 2^n}$, and a quantum state $|\psi\rangle \in \mathbb{C}^{2^n}$
Output: the quantum state $e^{itH}|\psi\rangle$

Unlike problems we have seen so far, this is a *quantum* problem, with *quantum* input and *quantum* output. Classical computers can also be used to solve a classical version of this problem, in which the input state $|\psi\rangle$ and output state $e^{itH}|\psi\rangle$ are not quantum states, but rather, classical descriptions of quantum states. As far as we know, the time required by a classical computer for this problem scales exponentially with n, whereas we will see in Section 6.5 that quantum computers can do much better, for example when H is s-sparse for $s = \text{poly}(n)$, the required resources scale polynomial in n. However, being able solve the classical version of this problem could potentially be much more useful. For example, if you want to know a specific entry of the vector describing the state $e^{itH}|\psi\rangle$, you cannot learn it with a single copy of the state! We will not consider these details in this course, leaving it up to the end user of a Hamiltonian simulation algorithm to figure out how to make use of this state, but it is important to understand this distinction.

6.2.2 Quantum Linear System Solvers

Another example of a quantum problem is the following, called the quantum linear system problem.

Problem: QLSP

Input: sparse access oracles $(\mathcal{O}_A, \mathcal{O}_r, \mathcal{O}_c)$ for a matrix $A \in \mathbb{C}^{2^n \times 2^n}$, and a quantum state $|\vec{b}\rangle = \frac{1}{\|\vec{b}\|}\sum_{i\in[2^n]} b_i|i\rangle \in \mathbb{C}^{2^n}$
Output: a quantum state $|\vec{x}\rangle = \frac{1}{\|\vec{x}\|}\sum_{i\in[2^n]} x_i|i\rangle \in \mathbb{C}^{2^n}$ such that $A\vec{x} = \vec{b}$, where we interpret $\vec{x}$ and $\vec{b}$ as column vectors in $\mathbb{C}^{2^n}$ (not necesarily normalized)

Strictly speaking, we do not need A to be square, but this is without loss of generality. Throughout these notes, we will assume A is invertible, but if it is not, the techniques mentioned here also apply, with the output $\vec{x}$ being $A^+\vec{b}$, where A^+ is the pseudoinverse. This is a quantum analogue of the following exremely important classical problem, the linear system problem.

Problem: LSP

Input: sparse access oracles $(\mathcal{O}_A, \mathcal{O}_r, \mathcal{O}_c)$ for a matrix $A \in \mathbb{C}^{2^n \times 2^n}$, and a vector $\vec{b} \in \mathbb{C}^{2^n}$, for which we have a full classical description, for example, an oracle $\mathcal{O}_{\vec{b}}$ that allows us to query the entries of $\vec{b}$
Output: a full classical description of $\vec{x} \in \mathbb{C}^{2^n}$ such that $A\vec{x} = \vec{b}$

Linear system solving is extremely important in a myriad of classical applications, including, to the delight of hypesters everywhere, many machine learning applications. While a classical computer for this problem needs $\Omega(2^n)$ time just to read the input and write the output, a quantum computer can solve the *quantum version* in poly(n) time if A is poly(n)-sparse and *well-conditioned* [HHL09], as we will see in Section 6.6. However, note the clear differences between these two problems that prevent a quantum algorithm for QLSP from being directly applicable to LSP:

Issue 1 Given a classical description of a vector, $\vec{b} \in \mathbb{C}^{2^n}$, even just reading it, in order to generate a corresponding quantum state $|\vec{b}\rangle$, would take time 2^n.

Issue 2 Given a quantum state $|\vec{x}\rangle$, the number of copies needed to get a full classical description of the vector $\vec{x}$ is exponential in n.

This was famously detailed in [Aar15], but by that point, there was so much hype around quantum linear systems solvers that it was hard to reverse.

Nonetheless, algorithms for QLSP are still potentially interesting. For example, an application where you only want to sample a non-zero entry of $\vec{b}$ would not suffer from Issue 2, and such an application was given in [KP17] for *recommender systems*, where the non-zero entries of some output vector represent recommendations (of products, tv shows, etc., based on data about known preferences). In addition, they tried to get around Issue 1 by describing a way to store data in a special classical data structure, written in QCRAM, which is memory that stores classical bits, but can be queried in superposition. They describe a data structure such that if a vector $\vec{v} \in \mathbb{C}^N$ is stored in this data structure (in QCRAM), then the superposition

$$\frac{1}{\|\vec{v}\|} \sum_{i\in[n]} v_i |i\rangle$$

can be prepared in $O(\log N)$ local gates and QCRAM reads. Note that actually preparing the data in such a data structure would require resources that scale like the size of the data set, but it is generally assumed that this has already been done (perhaps when the data was initially collected) and so this cost is not counted in an algorithm that uses the data as input. For details, see [KP17]. We will refer to this as a *QCRAM data structure*.

The quantum recommender system was an extremely exciting development, because it was the first proposal for a real application of a quantum linear system solver – that is, it was clearly worked out how the application would be useful even in light of Issues 1 and 2 – that seemed to get an exponential speedup over the best classical algorithms. Unfortunately, it turns out that the assumptions made about the input in order to get to a good quantum algorithm for recommender systems could also be leveraged to get a better *classical* algorithm, that is only polynomially worse than the quantum one, meaning there is no exponential speedup [Tan19]. This technique of "dequantization" has since been applied to a number of related problems with similar quantum algorithms. So while the quantum recommender system is still a very interesting algorithm, it also serves as a reminder that a thorough comparison with classical techniques is needed to truly establish a quantum speedup.

6.3 Block Encodings

Consider the following task:

> for a given matrix $A \in \mathbb{C}^{M\times N}$, and quantum state $|\psi\rangle \in \mathbb{C}^N$, generate the state $A|\psi\rangle$.

This is not a feasible task, because in general, A is not an isometry, and so $A|\psi\rangle$ could have arbitrary norm. What we could hope for is to take some matrix that is scaled so that $\|A\| \leq 1$, and then to unitarily map $|0\rangle|\psi\rangle$ to

$$|0\rangle(A|\psi\rangle) + |1\rangle|\tilde{\psi}\rangle, \tag{6.2}$$

for some $|\tilde{\psi}\rangle$ such that $\||A|\psi\rangle\|^2 + \left\||\tilde{\psi}\rangle\right\|^2 = 1$. Then, if we like, we can use amplitude amplification (see Theorem 2.2.11) to obtain a good approximation to the state:

$$\frac{A|\psi\rangle}{\|A|\psi\rangle\|}.$$

If we can generate a state like in (6.2) by some means (the "solution-sample" step in amplitude amplification), then we can easily reflect around states where the first qubit is $|0\rangle$ (the "check" step in amplitude amplification), and so we can approximate $A|\psi\rangle / \||A|\psi\rangle\|$ using $1/\|A|\psi\rangle\|$ repetitions.

We have already seen some examples of problems where we would like to be able to apply a matrix to a state.

Example 6.3.1 (Quantum linear system solver)**.** The goal of the quantum linear system problem is, given a state $|\vec{b}\rangle$, to output a state proportional to $A^{-1}|\vec{b}\rangle$, or if A is not invertible, $A^{+}|\vec{b}\rangle$, where A^{+} is the pseudoinverse of A.

We can restrict our attention to square (even Hermitian) matrices, as if A is not square, we can just use

$$\begin{bmatrix} 0 & A \\ A^\dagger & 0 \end{bmatrix}.$$

Block Encodings We have not yet said how we are "given" A, but here's a really naive idea: stick it in the top left corner of a unitary:

$$U = \begin{bmatrix} A & \cdot \\ \cdot & \cdot \end{bmatrix} = |0\rangle\langle 0| \otimes A + \dots.$$

This is a naive idea because it is not at all immediate how to implement a unitary U with this structure, but if we *could* implement U, then we could apply A to any state $|\psi\rangle$, by simply applying U to $|0\rangle|\psi\rangle$.

Of course, this is only possible if $\|A\| \leq 1$. Otherwise, we can scale A down by some α:

$$U = \begin{bmatrix} \frac{A}{\alpha} & \cdot \\ \cdot & \cdot \end{bmatrix}.$$

We call this a *block encoding* of A, and for even more flexibility, we allow block encodings to have some error.

Definition 6.3.2 (Block Encoding)**.** *For a unitary U on $\mathcal{H} = \mathcal{H}_0 \otimes \mathcal{H}_1$, $\varepsilon \in [0,1)$, and $\alpha \in \mathbb{R}_{>0}$, we say that U is an (α, ε)-block encoding of A if:*

$$\|A - \alpha(\langle 0|_{\mathcal{H}_0} \otimes I_{\mathcal{H}_1})U(|0\rangle_{\mathcal{H}_0} \otimes I_{\mathcal{H}_1})\| \leq \varepsilon.$$

If $\alpha = 1$ and $\varepsilon = 0$, this condition becomes

$$A = (\langle 0|_{\mathcal{H}_0} \otimes I_{\mathcal{H}_1})U(|0\rangle_{\mathcal{H}_0} \otimes I_{\mathcal{H}_1}),$$

and we just say U is a block encoding of A.

This idea is actually not so silly, as we now illustrate with some examples, and constructions.

Example 6.3.3. Any unitary U on $\mathcal{H} = \mathbb{C} \otimes \mathcal{H}$ is a block encoding of itself.

Example 6.3.4. Let U_0 and U_1 be unitaries on $\mathcal{H}_1$, and define U on $\mathcal{H} = \mathbb{C}^2 \otimes \mathcal{H}_1$ by:

$$U = |0\rangle\langle 0| \otimes U_0 + |1\rangle\langle 1| \otimes U_1.$$

Then U is a block encoding of U_0.

Example 6.3.5 (Random Walk Block Encoding)**.** Fix a graph G and let P be the transition matrix of a random walk on G. For any $u \in V(G)$, define:

$$|\bar{\psi}_\star(u)\rangle := \sum_{v \in V(G)} \sqrt{P_{u,v}}|u,v\rangle \quad \text{and} \quad |\bar{\psi}'_\star(u)\rangle := \sum_{v \in V(G)} \sqrt{P_{u,v}}|v,u\rangle.$$

Note that since for any $u \in V(G)$, $P_{u,v} = \frac{\mathsf{w}_{u,v}}{\mathsf{w}_u}$, the states $|\bar{\psi}_\star(u)\rangle$ are just normalizations of some version of the star states $|\psi_\star(u)\rangle$ that we saw in Section 3.7.1.[2] Recall that in order to reflect around star states, we needed to be able to generate them. Suppose we have unitaries that act, for all $u \in V(G)$, as:

$$U_{\mathcal{A}}|0,u\rangle = |\bar{\psi}_\star(u)\rangle \quad \text{and} \quad U_{\mathcal{B}}|0,u\rangle = |\bar{\psi}'_\star(u)\rangle.$$

Then $U_{\mathcal{A}}^\dagger U_{\mathcal{B}}$ is called the *quantum walk operator*, and it relates to what we have seen by:

$$(2\Pi_{\mathcal{A}} - I)(2\Pi_{\mathcal{B}} - I) = U_{\mathcal{A}}(2\Pi_0 - I)U_{\mathcal{A}}^\dagger U_{\mathcal{B}}(2\Pi_0 - I)U_{\mathcal{B}}^\dagger$$

where $\mathcal{A} = \mathrm{span}\{|\psi_\star(u)\rangle : u \in V(G)\}$, $\mathcal{B} = \mathrm{span}\{|\psi'_\star(u)\rangle : u \in V(G)\}$, and $\Pi_0 = |0\rangle\langle 0| \otimes \sum_{v\in V(G)} |v\rangle\langle v|$. Then we have

$$\begin{aligned}
(\langle 0|\otimes I)U_{\mathcal{A}}^\dagger U_{\mathcal{B}}(|0\rangle\otimes I) &= (\langle 0|\otimes I)\left(\sum_{u\in V(G)} |0,u\rangle\langle\bar{\psi}_\star(u)| \sum_{v\in V(G)} |\bar{\psi}'_\star(v)\rangle\langle 0,v|\right)(|0\rangle\otimes I)\\
&= \sum_{u,v\in V(G)} \langle\bar{\psi}_\star(u)|\bar{\psi}'_\star(v)\rangle |u\rangle\langle v|\\
&= \sum_{u,v\in V(G)} \sqrt{P_{u,v}P_{v,u}}|u\rangle\langle v|.
\end{aligned}$$

When P is symmetric, $\sqrt{P_{u,v}P_{v,u}} = P_{u,v}$, and this tells us that $U_{\mathcal{A}}^\dagger U_{\mathcal{B}}$ is a block encoding of P. More generally, by detailed balance (see Section 3.2) we have $\pi_u P_{u,v} = \pi_v P_{v,u}$, so

$$\sqrt{P_{u,v}P_{v,u}} = \sqrt{\frac{\pi_u}{\pi_v}P_{u,v}},$$

and $U_{\mathcal{A}}^\dagger U_{\mathcal{B}}$ block encodes the *discriminant* of P, defined

$$D(P) = \sum_{u,v\in V(G)} \sqrt{\frac{\pi_u}{\pi_v}P_{u,v}}|u\rangle\langle v| = \mathsf{diag}(\pi)^{1/2}P\mathsf{diag}(\pi)^{-1/2},$$

where $\mathsf{diag}(\pi) = \sum_{u\in V(G)} \pi_u|u\rangle\langle u|$. Since $\mathsf{diag}(\pi)^{1/2}\mathsf{diag}(\pi)^{-1/2} = I$, $D(P)$ is similar to P. In early quantum walk papers, such as [Sze04, MNRS11], the spectrum of $D(P)$, which is the same as the spectrum of P, was used to say something about the phases of the unitary $U_{\mathcal{A}}^\dagger U_{\mathcal{B}}$.

This last example shows that we can implement block encodings of certain matrices when we can generate related vectors. The general case of this is captured in the following lemma.

Lemma 6.3.6. *A* Gram matrix *is a matrix of the form $\sum_{i,j\in[N]} \langle\psi_i|\phi_j\rangle|i\rangle\langle j|$ for some sets of vectors $\{|\psi_i\rangle\}_{i\in[N]}$ and $\{|\phi_j\rangle\}_{j\in[N]}$. Suppose A is the Gram matrix of orthonormal sets of vectors $\{|\psi_i\rangle\}_{i\in[N]}$ and $\{|\phi_j\rangle\}_{j\in[N]}$, and let U_L and U_R be a unitaries that generate them, meaning that for all $i,j \in [N]$,*

$$U_L : |0,i\rangle \mapsto |\psi_i\rangle \quad \textit{and} \quad U_R : |0,j\rangle \mapsto |\phi_j\rangle.$$

Then $U_L^\dagger U_R$ is a block encoding of A.

Using this lemma, we can implement a block encoding of A given sparse access oracles for A, which means that it is at least as natural to suppose we have a block encoding of A as it is to suppose we have oracle access.

[2]The correspondance with the star states we saw is the following. One way of making G bipartite is to make two copies of G, an $\mathcal{A}$ copy and a $\mathcal{B}$ copy, and for every edge $\{u,v\}$ in G add an edge from the copy of u in $\mathcal{A}$ to the copy of v in $\mathcal{B}$, and call it (u,v); and also add an edge from the copy of v in $\mathcal{A}$ to the copy of u in $\mathcal{B}$, and call that (v,u). This is essentially like adding a bit (indicating $\mathcal{A}$ copy or $\mathcal{B}$ copy) to the description of each vertex, and doing a walk on G expect that we flip that bit at every step.

Exercise 6.3.1 (Block Encoding from Sparse Access)**.** *Let $(\mathcal{O}_A, \mathcal{O}_r, \mathcal{O}_c)$ be sparse access oracles for $A \in \mathbb{C}^{2^n \times 2^n}$ (see Definition 6.2.1). Show how to implement an $(s, 0)$-block encoding of A, U on $\mathbb{C}^{2^a} \otimes \mathbb{C}^{2^n}$ for some $a = O(n)$, using $O(1)$ calls to these oracles. Hint: You may use the fact that you can implement a unitary of the form $|0\rangle|A_{i,j}\rangle \mapsto \left(A_{i,j}|0\rangle + \sqrt{1-|A_{i,j}|^2}|1\rangle\right)|A_{i,j}\rangle$ using 0 queries (having already queried $A_{i,j}$).*

The sparse access model assumes we can compute the entries of A, which suggests that they are somehow structured. This would not generally be true if A is made up of some real world *data*, as in many data processing applications that use linear algebra techniques (not to mention the fact that data is often not sparse). As we mentioned at the end of Section 6.2.2, it is possible to store a vector $\vec{v} \in \mathbb{C}^N$, possibly corresponding to some data collected in the wild, in a particular classical data structure, in QCRAM, in such a way that generting a quantum state proportional to $\sum_{i\in[N]} v_i|i\rangle$ can be done in $O(\log N)$ local gates and QCRAM reads (see [KP17] for details). Given such data structures storing each column of A (or alternatively, each row of A), we can also implement a block encoding of A, as the following exercise shows.

Exercise 6.3.2. *Fix $A \in \mathbb{C}^{N\times N}$ and suppose you can implement a unitary that acts, for every $j \in [N]$, as:*

$$U_c|0,j\rangle = |0\rangle \sum_{i\in[N]} \frac{A_{i,j}}{\|A\|}|i,j\rangle + |1\rangle|\tilde{\psi}_j\rangle$$

for some $|\tilde{\psi}_j\rangle$ – this is possible because $\|A\|$ is always an upper bound on the norm of any column of A. Show how to implement a $(\|A\|, 0)$-block encoding of A using a single call to U_r (and a modest number of local gates).

This makes block encodings a rather general input model, since we can build them from various different input models, meaning algorithms that work given a block encoding will also apply to these other input models. However, this doesn't help us when we want to do something like quantum linear system solving, as usually then we assume we are given A in some natural way (sparse access, for example) rather than A^{-1}, which is what we want to apply to the state. However, it turns out that block-encodings lend themselves well to transformations. We first mention some simple examples, before seeing more general transformations in Section 6.4.

Modifying Block Encodings Suppose U on $\mathcal{H} = \mathcal{H}_0 \otimes \mathcal{H}_1$ is a block encoding of a matrix A on $\mathcal{H}_1$, and V on $\mathcal{H}' = \mathcal{H}'_0 \otimes \mathcal{H}_1$ is a block encoding of a matrix B, also acting on $\mathcal{H}_1$. For clarity, let us swap the order of the tensored spaces in $\mathcal{H}'$ so that $\mathcal{H}' = \mathcal{H}_1 \otimes \mathcal{H}'_0$, and we have:

$$(\langle 0|_{\mathcal{H}_0} \otimes I_{\mathcal{H}_1})U(|0\rangle_{\mathcal{H}_0} \otimes I_{\mathcal{H}_1}) = A \quad \text{and} \quad (I_{\mathcal{H}_1} \otimes \langle 0|_{\mathcal{H}'_0})V(I_{\mathcal{H}_1} \otimes |0\rangle_{\mathcal{H}'_0}) = B.$$

Then we claim that $(U \otimes I_{\mathcal{H}'_0})(I_{\mathcal{H}_0} \otimes V)$, which is a unitary on $\mathcal{H}_0 \otimes \mathcal{H}_1 \otimes \mathcal{H}'_0$, is a block encoding of AB (up to reordering of the tensored spaces):

$$\begin{aligned}
&(\langle 0|_{\mathcal{H}_0} \otimes I_{\mathcal{H}_1} \otimes \langle 0|_{\mathcal{H}'_0})(U \otimes I_{\mathcal{H}'_0})(I_{\mathcal{H}_0} \otimes V)(|0\rangle_{\mathcal{H}_0} \otimes I_{\mathcal{H}_1} \otimes |0\rangle_{\mathcal{H}'_0}) \\
&= (((\langle 0|_{\mathcal{H}_0} \otimes I_{\mathcal{H}_1})U \otimes \langle 0|_{\mathcal{H}'_0})(|0\rangle_{\mathcal{H}_0} \otimes V(I_{\mathcal{H}_1} \otimes |0\rangle_{\mathcal{H}'_0})) \\
&= (((\langle 0|_{\mathcal{H}_0} \otimes I_{\mathcal{H}_1})U \otimes 1)(|0\rangle_{\mathcal{H}_0} \otimes I_{\mathcal{H}_1} \otimes \langle 0|_{\mathcal{H}'_0})(1 \otimes V(I_{\mathcal{H}_1} \otimes |0\rangle_{\mathcal{H}'_0})) \\
&= (((\langle 0|_{\mathcal{H}_0} \otimes I_{\mathcal{H}_1})U)(|0\rangle_{\mathcal{H}_0} \otimes I_{\mathcal{H}_1})(I_{\mathcal{H}_1} \otimes \langle 0|_{\mathcal{H}'_0})(V(I_{\mathcal{H}_1} \otimes |0\rangle_{\mathcal{H}'_0})) \\
&= AB.
\end{aligned}$$

If we reorder the spaces to $(\mathcal{H}_0 \otimes \mathcal{H}'_0) \otimes \mathcal{H}_1$, we get the usual notion of block encoding.

Exercise 6.3.3. *Let U and V be unitaries acting on the same space, $\mathcal{H} = \mathcal{H}_0 \otimes \mathcal{H}_1$, and suppose U is a block encoding of a matrix A acting on $\mathcal{H}_1$, and V is a block encoding of B acting on $\mathcal{H}_1$. Show that the unitary U' in Figure 6.1 is a $(2, 0)$-block encoding of $A + B$.*

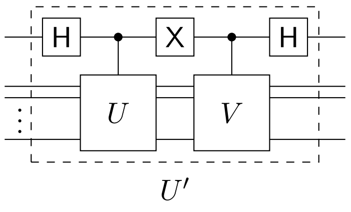

Figure 6.1: A circuit for adding block encodings. Call the resulting unitary (everything inside the dashed box) U'.

Remark 6.3.7. *The definition of a block encoding is basis dependent, but most of what we say about block encodings in these notes can be extended to* projected unitary encodings*, which are unitaries U such that $\tilde{\Pi}U\Pi = A$, for some projectors Π and $\tilde{\Pi}$. See [Gil19] for details.*

6.4 The Quantum Singular Value Transform

6.4.1 Quantum Signal Processing

Fix a "signal" rotation, that is a function of x on the domain $[-1, 1]$:

$$W(x) := \begin{bmatrix} x & i\sqrt{1-x^2} \\ i\sqrt{1-x^2} & x \end{bmatrix}.$$

Let us think of this as an input oracle, through which we access the single variable x (and in fact, it's a block encoding of the 1-by-1 matrix $[x]$). *Signal processing* takes this unitary and composes it with unitaries of the form

$$S(\phi) := e^{i\phi \mathsf{Z}} = \begin{bmatrix} e^{i\phi} & 0 \\ 0 & e^{-i\phi} \end{bmatrix} = e^{i\phi} \begin{bmatrix} 1 & 0 \\ 0 & e^{-2i\phi} \end{bmatrix} \tag{6.3}$$

for some $\phi \in \mathbb{R}$. We can think of this as building up a single-qubit quantum circuit using queries to $W(x)$, and gates of the form (6.3), for any chosen values of ϕ. Concretely, for any $\vec{\phi} = (\phi_0, \dots, \phi_T) \in \mathbb{R}^{T+1}$, we let

$$U_{\vec{\phi}}(x) := S(\phi_0)W(x)S(\phi_1)W(x)S(\phi_2)\dots S(\phi_{T-1})W(x)S(\phi_T).$$

Then let

$$p_{\vec{\phi}}(x) := |\langle 0|U_{\vec{\phi}}(x)|0\rangle|^2,$$

which is the probability of measuring 0 if we apply $U_{\vec{\phi}}(x)$ to $|0\rangle$. Continuing to think of $U_{\vec{\phi}}(x)$ as a single-qubit circuit, or algorithm, with input $|0\rangle$, and now also thinking of $|0\rangle$ as the unique *accepting* state, $p_{\vec{\phi}}(x)$ is the probability of accepting on input x.

Example 6.4.1. Suppose $x \in \{0, 1\}$. Then $W(x) = (i\mathsf{X})^{1-x}$, so we can query the bit value of $1 - x$ using a single application of $W(x)$. If we let $\vec{\phi} = (\phi_0, \phi_1) = (0, 0)$, we get $U_{\vec{\phi}}(x) = W(x) = (i\mathsf{X})^{1-x}$, and $p_{\vec{\phi}}(x) = 1$ if $x = 1$, and $p_{\vec{\phi}}(x) = 0$ if $x = 0$. In fact, with this setting of $\vec{\phi}$, $p_{\vec{\phi}}(x) = x^2$ for any $x \in [-1, 1]$.

On the other hand, suppose $x = \left(1 - \frac{1}{\sqrt{2}}\right) b + \frac{1}{\sqrt{2}}$ for some unknown $b \in \{0, 1\}$. Then this is a "weaker" signal, so we will need more applications of $W(x)$ to learn b. For these particular parameters, 2 applications suffice. Choosing $\phi_0 = \phi_1 = \phi_2 = 0$, we have:

$$\begin{aligned} p_{\vec{\phi}}(x) &= \left|\langle 0|W(x)^2|0\rangle\right|^2 = \begin{bmatrix} x & i\sqrt{1-x^2} \end{bmatrix} \begin{bmatrix} x \\ i\sqrt{1-x^2} \end{bmatrix} \\ &= x^2 - (1 - x^2) = 2x^2 - 1 = 2\left(\left(1 - \frac{1}{\sqrt{2}}\right) b + \frac{1}{\sqrt{2}}\right)^2 - 1 = b \end{aligned}$$

for $b \in \{0, 1\}$, so we can perfectly learn b with 2 uses of $W(x)$.

Exercise 6.4.1. *Chebyshev polynomials of the first kind are defined by the equations, for $d \in \mathbb{Z}_{\geq 0}$, $\mathsf{T}_d(\cos\theta) = \cos(d\theta)$. Equivalently, they can be defined by the recurrence relation:*

$$\begin{aligned} \mathsf{T}_0(x) &= 1 \\ \mathsf{T}_1(x) &= x \\ \forall d \geq 1,\ \mathsf{T}_{d+1}(x) &= 2x\mathsf{T}_d(x) - \mathsf{T}_{d-1}(x). \end{aligned}$$

Chebyshev polynomials of the second kind are defined by the equations $\mathsf{U}_d(\cos\theta)\sin\theta = \sin((d+1)\theta)$. Equivalently, they can be defined by the recurrence relation:

$$\begin{aligned} \mathsf{U}_0(x) &= 1 \\ \mathsf{U}_1(x) &= 2x \\ \forall d \geq 1,\ \mathsf{U}_{d+1}(x) &= 2x\mathsf{U}_d(x) - \mathsf{U}_{d-1}(x). \end{aligned}$$

Prove that for any $d \geq 0$, $\langle 0|W(x)^d|0\rangle = \mathsf{T}_d(x)$, by showing the stronger statement: for all $d \geq 1$,

$$W(x)^d = \begin{bmatrix} \mathsf{T}_d(x) & i\mathsf{U}_{d-1}(x)\sqrt{1-x^2} \\ i\mathsf{U}_{d-1}(x)\sqrt{1-x^2} & \mathsf{T}_d(x) \end{bmatrix}.$$

Thus, if $\vec{\phi} \in \mathbb{C}^{d+1}$ is all 0s, $p_{\vec{\phi}}(x) = \mathsf{T}_d(x)$. What other functions of x can we "compute" this way? The answer is any polynomial satisfying certain minor constraints. Recall that $\mathbb{C}[x]$ is the set of polynomials in x with complex coefficients, and for any $P \in \mathbb{C}[x]$, let P^* be obtained by taking the complex conjugate of each coefficient. Recall that a polynomial is even if and only if it only has terms of even degree, and a polynomial is odd if and only if it only has terms of odd degree. Then we have the following.

Theorem 6.4.2 (Quantum Signal Processing)**.** *For any $\vec{\phi} \in \mathbb{R}^{T+1}$,*

$$U_{\vec{\phi}}(x) = \begin{bmatrix} P(x) & iQ(x)\sqrt{1-x^2} \\ iQ^*(x)\sqrt{1-x^2} & P^*(x) \end{bmatrix} \tag{6.4}$$

for polynomials $P, Q \in \mathbb{C}[x]$ such that

1. $\deg(P) \leq T$ *and* $\deg(Q) \leq T-1$;
2. *P is even if and only if T is even, and otherwise it is odd; and Q is even if and only if T is odd, and otherwise it is odd;*
3. *for all $x \in [-1,1]$, $|P(x)|^2 + (1-x^2)|Q(x)|^2 = 1$.*

Furthermore, for any polynomials $P, Q \in \mathbb{C}[x]$ satisfying the above properties, there exists a sequence $\phi \in \mathbb{R}^{T+1}$ satisfying (6.4). Finally, for any polynomial P of degree d, there exists a polynomial Q such that P and Q satisfy conditions 1-3 if and only if:

A. *P has the same parity as d.*

B. *For all $x \in [-1,1]$, $|P(x)| \leq 1$.*

C. *For all $x \in (-\infty,-1] \cup [1,\infty)$, $|P(x)| \geq 1$.*

D. *If d is even, then for all $x \in \mathbb{R}$, $P(ix)P^*(ix) \geq 1$.*

Thus, for some appropriate choice of $\vec{\phi}$, we can apply any polynomial P to x as long as it satsifies the conditions A-D. The cost of this construction in terms of queries to $W(x)$ is the degree of P. We can further extend the class of polynomials achievable by noting that by Exercise 6.3.3, we can also *add* block encodings, and a 2-by-2 matrix with $P(x)$ in the top-left corner is a special case of a block

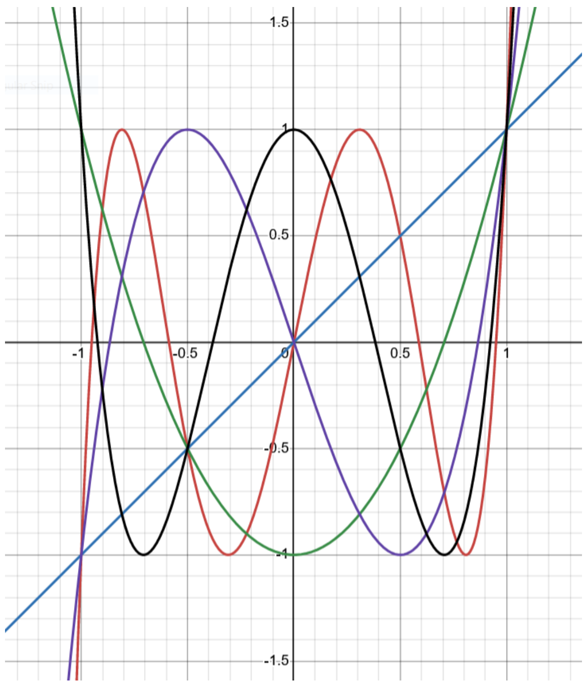

Figure 6.2: We already know that Chebyshev polynomials of the first kind satisfy conditions A-D. Pictured is $\mathsf{T}_d(x)$ for $d = 1, \dots, 5$. Chebyshev polynomials of the first kind form a basis for the space of polynomials, so, in particular, any polynomial of degree d can be expressed as a linear combination of $\mathsf{T}_0, \dots, \mathsf{T}_d$.

encoding. For example, while Theorem 6.4.2 requires P to be even or odd, any polynomial is the sum of an even polynomial and an odd polynomial. As a second example, we know that Chebyshev polynomials satisfy the conditions A-D, and any polynomial of degree d can be expressed as a linear combination of Chebyshev polynomials of degree at most d.

Remark 6.4.3. *For any real polynomial, $R \in \mathbb{R}[x]$, of degree d, with parity d, and such that for all $x \in [-1, 1]$, $|R(x)| \leq 1$, there exists a $P \in \mathbb{C}[x]$ whose* real *part is R. (See [Gil19, Corollary 2.2.8]).*

We will not prove Theorem 6.4.2, but you will show part of the easier direction as an exercise.

Exercise 6.4.2. *Fix any $\vec{\phi} \in \mathbb{R}^{T+1}$, and define $P(x) = \langle 0|U_{\vec{\phi}}(x)|0\rangle$. Prove, by induction on $T \geq 1$ that $P(x)$ is a polynomial (in x) of degree T.*

6.4.2 Generalization to Block Encodings

In the previous section, we saw how, for any polynomial P satsifying certain conditions, given query access to a 2-by-2 unitary $W(x)$ with a real number x of magnitude at most 1 in the top-left corner, we could implement a 2-by-2 unitary with $P(x)$ in the top-left corner using applications of $W(x)$ alternating with some other rotations. In this section, we will generalize these results to show how, given query access to a unitary W_X with some *matrix* X in the top-left corner (i.e., a block encoding of X), we can implement a unitary with $P(X)$ in the top-left corner (i.e. a block encoding of $P(X)$) using applications of W_X alternating with some other rotations.

In analogy with $S(\phi)$ in (6.3), for $\phi \in \mathbb{R}$, and some implicit space $\mathcal{H} = \mathcal{H}_0 \otimes \mathcal{H}_1$, define

$$S_{\Pi_0}(\phi) = e^{i\phi(2\Pi_0 - I)} = e^{i\phi}\Pi_0 + e^{-i\phi}(I - \Pi_0), \tag{6.5}$$

on $\mathcal{H}$, where $\Pi_0 = |0\rangle\langle 0|_{\mathcal{H}_0} \otimes I_{\mathcal{H}_1}$. For a square matrix X on $\mathcal{H}_1$, and a block encoding of X, W_X on

$\mathcal{H}$, and $\vec{\phi} \in \mathbb{R}^{T+1}$, define

$$U_{\vec{\phi}}(X) := \begin{cases} S_\Pi(\phi_0)W_X^\dagger S_\Pi(\phi_1)W_X S_\pi(\phi_2)W_X^\dagger \dots S_\Pi(\phi_{n-1})W_X^\dagger S_\Pi(\phi_T)W_X & \text{if } T \text{ is even} \\ S_\Pi(\phi_0)W_X S_\Pi(\phi_1)W_X^\dagger S_\pi(\phi_2)W_X \dots S_\Pi(\phi_{n-1})W_X^\dagger S_\Pi(\phi_T)W_X & \text{if } T \text{ is odd} \end{cases}$$

We stress that unlike in Section 6.4.1, where $W(x)$ is a function of x, there are many possible block encodings W_X for X, so $U_{\vec{\phi}}(X)$ is actually a function of W_X rather than just X.

Then we have the following generalization of Theorem 6.4.2.

Theorem 6.4.4 ([GSLW19])**.** *Let P be a degree d polynomial satisfying conditions A-D of Theorem 6.4.2. Then there exists $\vec{\phi} \in \mathbb{R}^{d+1}$ (in fact, the same one as in Theorem 6.4.2) such that $U_{\vec{\phi}}(X)$ is a block encoding of $P(X)$.*

What Theorem 6.4.4 says is that if we are able to implement a block encoding for X, then we can use T calls to it (or its inverse), as well as $T+1$ basic rotations, to implement a block encoding of $P(X)$, for any degree T polynomial $P(\cdot)$ satsifying certain conditions. We can further extend this to any polynomial (up to some scaling) by using the fact that we can add block encodings (Exercise 6.3.3). This is especially powerful, considering that *any* smooth function can be approximated by a polynomial of sufficiently high degree.

To give some idea of why the 2-dimensional "qubit" case in Theorem 6.4.2 extends to higher dimensions to get Theorem 6.4.4, we consider the special case of the following unitary, which is a block encoding of some Hermitian H with $\|H\| \leq 1$:

$$W(H) = |0\rangle\langle 0| \otimes H + i|0\rangle\langle 1| \otimes \sqrt{I-H^2} + i|1\rangle\langle 0| \otimes \sqrt{I-H^2} + |1\rangle\langle 1| \otimes H = \begin{bmatrix} H & i\sqrt{I-H^2} \\ i\sqrt{I-H^2} & H \end{bmatrix}.$$

Let $H = \sum_{j\in\mathcal{J}} \lambda_j \Pi_j$. Then

$$\begin{aligned} W(H) &= \sum_{j\in\mathcal{J}} \left(|0\rangle\langle 0| \otimes \lambda_j \Pi_j + i|0\rangle\langle 1| \otimes \sqrt{1-\lambda_j^2}\Pi_j + i|1\rangle\langle 0| \otimes \sqrt{1-\lambda_j^2}\Pi_j + |1\rangle\langle 1|\lambda_j\Pi_j \right) \\ &= \sum_{j\in\mathcal{J}} \underbrace{\begin{bmatrix} \lambda_j & i\sqrt{1-\lambda_j^2} \\ i\sqrt{1-\lambda_j^2} & \lambda_j \end{bmatrix}}_{W(\lambda_j)} \otimes \Pi_j. \end{aligned}$$

Thus, $W(H)$ is a direct sum of matrices of the form $W(\lambda_j)$, where λ_j are the eigenvalues of H. For this special case, the decomposition of $\mathcal{H}$, the space acted on by $W(H)$, is $\mathcal{H} = \mathcal{H}_0 \otimes \mathcal{H}_1$ where $\mathcal{H}_0 = \mathbb{C}^2$ and $\mathcal{H}_1$ is the space acted on by H, so we have

$$\Pi_0 = |0\rangle\langle 0|_{\mathcal{H}_0} \otimes I_{\mathcal{H}_1} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \otimes \sum_{j\in\mathcal{J}} \Pi_j$$

and

$$I - \Pi_0 = |1\rangle\langle 1|_{\mathcal{H}_0} \otimes I_{\mathcal{H}_1} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \otimes \sum_{j\in\mathcal{J}} \Pi_j$$

so

$$S_{\Pi_0}(\phi) = e^{i\phi}\Pi_0 + e^{-i\phi}(I - \Pi_0) = \sum_{j\in\mathcal{J}} \underbrace{\begin{bmatrix} e^{i\phi} & 0 \\ 0 & e^{-i\phi} \end{bmatrix}}_{S(\phi)} \otimes \Pi_j.$$

This is a special case of a more general phenomena that follows from a result called Jordan's lemma. This allows us to understand the structure of a general block encoding as direct sum of "qubit" block encodings, which is sometimes referred to as *qubitization*.

Example: Grover's algorithm For $x \in \{0,1\}^N$, with $N = 2^n$, let $\mathcal{G}$ be the Grover iterate from Section 2.2.1,

$$\mathcal{G} = \mathsf{H}^{\otimes n} R_0 \mathsf{H}^{\otimes n} \mathcal{O}_x^{\pm},$$

where we recall that $R_0 = 2|0\rangle\langle 0| - I$. Recall also that $|\pi\rangle = \mathsf{H}^{\otimes n}|0\rangle$. Let

$$\tilde{\mathcal{G}} := \mathsf{H}^{\otimes n} \mathcal{G} \mathsf{H}^{\otimes n} = R_0 \mathsf{H}^{\otimes n} \mathcal{O}_x^{\pm} \mathsf{H}^{\otimes n}.$$

We saw in Section 2.2.1 how to analyze a version of Grover's algorithm that consists of phase estimation of this unitary, but the more basic version of Grover's algorithm consists of measuring

$$\mathcal{G}^T|\pi\rangle = \mathsf{H}^{\otimes n} \tilde{\mathcal{G}}^T|0\rangle$$

for some appropriate T, in the computational basis, and checking if the result is marked.

Exercise 6.4.3. 1. *Show that $\tilde{\mathcal{G}}$ is a block encoding of the 1-by-1 matrix $[1 - 2|x|/N]$.*

2. *Just as we saw in Exercise 6.4.1 that $\langle 0|W(x)^d|0\rangle = \mathsf{T}_d(x)$, where $\mathsf{T}_d(x)$ is the d-th Chebyshev polynomial of the first kind, it turns out that for any block encoding U of a square matrix A, U^d is a block encoding of $\mathsf{T}_d(A)$. Show that for any d, if $|x| = 0$, measuring $\tilde{\mathcal{G}}^d|0\rangle$ yields the all 0s string with probability 1.*

3. *Suppose $|x|/N \leq 1/2$. Show that if you know $|x|$ exactly, there is a choice of $d = O(\sqrt{N/|x|})$ such that measuring $\tilde{\mathcal{G}}^d|0\rangle$ yields the all 0s string with probability at most $1/2$.*

4. *Describe an algorithm that decides if $|x| = 0$ or not with one-sided error (your algorithm will have no error if $|x| = 0$) using measurements of $\tilde{\mathcal{G}}^d|0\rangle$ for appropriate choices of d, with worst-case running time $O(\sqrt{N})$.*

6.4.3 Example: Quantum Fast-Forwarding

Let P be the transition matrix of a random walk, and suppose, for simplicity, that P is symmetric (for example, it is a random walk on a regular graph). Then for a distribution σ, which is an ℓ_1-normalized real vector with non-negative entries, we have that[3] $P^t\sigma$ is the distribution after t steps of the random walk, starting from σ. Sampling from this distribution requires, in general, t applications of P. Can we speed this up with a quantum computer? In some cases. Consider the following quantum version of this task:[4]

> Given a state $|\psi\rangle \in \mathrm{span}\{|u\rangle : u \in V(G)\}$ and $t \in \mathbb{Z}_{\geq 0}$, generate a state of the form:
>
> $$|0\rangle P^t|\psi\rangle + |1\rangle|\tilde{\psi}\rangle.$$

Let us briefly discuss the use of generating such a state. It does *not* let us sample from the distribution one would obtain after t steps of a classical walk starting from some state. First, if we measure, we only obtain a result from the $P^t|\psi\rangle$ part of the state with probability $\left\|P^t|\psi\rangle\right\|^2$, which could be much smaller than 1, for example if $|\psi\rangle = |s\rangle$ for some $s \in V(G)$, and $P^t s$ is very spread out (as it is expected to be if t is not too small). And then even if we do get a sample from this part of the state, it will be distributed as $\Pr[u] \propto |\langle u|P^t|s\rangle|^2$ rather than $\langle u|P^t|s\rangle$. Nonetheless, this technique has found applications.

Example 6.4.5. In [AGJK20], it was shown that if t is at least some constant multiple of $\mathcal{HT}(P, M)$, measuring $P^t|\pi\rangle$ (for some modification of P that has self-loops on all marked vertices) yields a

[3]Because of how we defined transition matrices in Section 3.2, in general we apply P by right-multiplication: σP^t. Since P is symmetric, it doesn't matter, although we are abusing notation by letting σ denote either a row or a column vector – it should be clear from context which we mean.

[4]When P is not symmetric, we want to apply $D(P)^t$, where $D(P)$ is the *discriminant*, defined in Example 6.3.5.

marked vertex with reasonably high probability – and something similar holds for arbitrary initial states $|\sigma\rangle$ [AGJ20], yielding an alternative algorithm to the one we saw in Section 3.7.1, which is actually even stronger, since it *finds* a marked vertex, rather than just detecting one.

Example 6.4.6. Suppose you want to know, for a pair of vertices s and t in G, not just if they are connected, but if they are strongly connected. Concretely, imagine G consists of two or more strongly connected subgraphs, with only a small number of edges between these subgraphs, and you want to know if s and t are in the same or different subgraphs. The strategy is to generate $P^k|s\rangle$ and $P^k|t\rangle$ for some k that is sufficiently large so that $P^k|s\rangle$ will be fairly spread out across the subgraph containing s, and similarly for $P^k|t\rangle$. If they are in the same subgraph, these states will be highly overlapping, which can be detected through a procedure called a *swap test*.

The problem was first solved without the Quantum Singular Value Transform [AS19], but it can be easily described within the QSVT framework. If we have a block encoding of P, like the one in Example 6.3.5, then we can use Theorem 6.4.4 to get a block encoding of P^t that uses t calls to the block encoding for P. However, it turns out we can do better, because we can *approximate* the polynomial P^t using a polynomial of degree much less than t. The Chebyshev polynomials of the first kind, $\mathsf{T}_d(x)$, introduced in Exercise 6.4.1, form a basis for the space of polynomials, and so any polynomial of degree $\leq d$ can be expressed as a linear combination of $\{\mathsf{T}_0(x), \dots, \mathsf{T}_d(x)\}$. In particular, we have:

$$x^t = \sum_{k=0}^{t} p_{t,k}\mathsf{T}_k(x), \text{ where } p_{t,k} = \begin{cases} 2^{1-t}\binom{t}{\frac{t-k}{2}} & \text{if } k = t \mod 2 \text{ and } k > 0 \\ 2^{-t}\binom{t}{\frac{t}{2}} & \text{if } k = 0 \text{ and } t = 0 \mod 2 \\ 0 & \text{else.} \end{cases}$$

While x^t is, of course, a degree t polynomial, if we truncate the above sum at $\sqrt{ct}$ for some constant c, we can get a $\sqrt{ct}$-degree *approximation* of x^t, which is precisely what we need. Specifically, [AS19, Lemma 3] show that for any $t \geq 0$, $\varepsilon \in (0,1)$ and $c \geq 2\ln(2/\varepsilon)$, for all $x \in [-1,1]$:

$$\left| x^t - \sum_{k=0}^{\sqrt{ct}} p_{t,k}\mathsf{T}_k(x) \right| \leq \varepsilon.$$

This leads to the following.

Theorem 6.4.7. *For any ε, there is a $(1,\varepsilon)$-block encoding of P^t that can be implemented using $O(\sqrt{t\log(1/\varepsilon)})$ calls to the walk operator defined in Example 6.3.5.*

Applying this to any $|0\rangle|\psi\rangle$ yields $|0\rangle P^t|\psi\rangle + |1\rangle|\tilde{\psi}\rangle$, as desired.

6.5 Application to Hamiltonian Simulation

To apply what we have seen to the problem of Hamiltonian simulation, we want to use a block encoding of H (obtained from sparse access to H, or some other means) to implement a block encoding of e^{itH}. The function $f(x) = e^{itx}$ is not a polynomial, but we can get a pretty good approximation by truncating its Taylor series expansion:

$$e^{itx} = \sum_{k=0}^{\infty} \frac{(itx)^k}{k!}.$$

In fact, if we start from an $(\alpha,0)$-blocking encoding of H, the function we actually want to apply is $e^{it\alpha x}$, obtained by replacing t with αt.

Exercise 6.5.1. *1. Show that for any $q \geq et\alpha$, and any $x \in [-1,1]$,*

$$\left| e^{it\alpha x} - \sum_{k=0}^{q-1} \frac{(it\alpha x)^k}{k!} \right| \leq \frac{1}{1-1/e}\frac{(t\alpha)^q}{q!}.$$

2. *Use Stirling's approximation to conclude that for any* $q \geq \max\{2et\alpha, \log(1/\varepsilon)\}$,

$$\left| e^{it\alpha x} - \sum_{k=0}^{q-1} \frac{(it\alpha x)^k}{k!} \right| \leq \varepsilon.$$

From this, we can show the following.

Theorem 6.5.1 ([LC19]). *Let U on $\mathbb{C}^{2^a} \otimes \mathbb{C}^{2^n}$ be an $(\alpha, 0)$-block encoding of a Hermitian matrix $H \in \mathbb{C}^{2^n \times 2^n}$. Then we can implement a $(1, \varepsilon)$-block encoding of e^{itH} using $O(\alpha t + \log(1/\varepsilon))$ calls to U, and $O(a(\alpha t + \log(1/\varepsilon)))$ local gates.*[5]

For example, this means, by Exercise 6.3.1, that we can simulate a Hamiltonian that is s-sparse in $O(n(st + \log(1/\varepsilon)))$ local gates and calls to sparse access oracles. Note that it does not mean we can simulate *any* Hamiltonian $H \in \mathbb{C}^{2^n \times 2^n}$ with $\|H\| \leq 1$ in poly(n) resources – in fact, by a counting argument, most Hamilltonians require quantum circuits of size $2^{\Omega(n)}$ to simulate. However, the special case of sparse Hamiltonians, for example, $s = \text{poly}(n)$, is easy for quantum computers, but still appears hard for classical computers.

Remark 6.5.2. *The factor of a in the number of local gates comes from the cost of implementing the intermediate rotations $S_{\Pi_0}(\phi)$ in $U_{\vec{\phi}}(H)$, each of which can be reduced to implementing $2\Pi_0 - I$. Since $\Pi_0 = |0^a\rangle\langle 0^a| \otimes I$, this can be implemented by checking that each of the first a qubits is 0, in cost a. We have given most complexities in these notes in terms of the number of calls to a block encoding of the input, and omitted the number of local gates, but the number of local gates in our other constructions could also be recovered by taking the number of calls to the block encoding of the input and multiplying it by $a = \log \dim \mathcal{H}_0$.*

6.6 Application to Quantum Linear System Solving

For linear system solving, our aim will be to get a block encoding of A^{-1} from a block encoding of A. If we try to approximate the function $f(x) = x^{-1}$ by a polynomial on the interval $[-1, 1]$, we notice that it has a discontinuity at 0. In general, the complexity of inverting a matrix A scales with its *condition number*

$$\kappa = \|A\| \left\|A^{-1}\right\| = \frac{\max_{j \in \mathcal{J}} |\lambda_j|}{\min_{j \in \mathcal{J}} |\lambda_j|}$$

if A is normal with spectrum $\{\lambda_j\}_{j \in \mathcal{J}}$. If A has condition number κ, and $\|A\| \leq 1$ (which we will assume) then the eigenvalues of $A/\|A\| = A$ lie in the set $D = [-1, -1/\kappa] \cup [1/\kappa, 1]$, and so we only need to approximate f on that interval. Another issue is that $f(x) = x^{-1}$ has values in the range $[-\kappa, \kappa]$ on D, so in order to apply Theorem 6.4.4, we will instead approximate $\frac{1}{2\kappa} x^{-1}$, which will then give a block encoding of $\frac{1}{2\kappa} A^{-1}$.

Claim 6.6.1 ([GSLW19, MRTC21]). *There exists a degree $d = O(\kappa \log(\kappa/\varepsilon))$ polynomial P that $\varepsilon/2\kappa$-approximates $f(x) = \frac{1}{2\kappa} x^{-1}$ on $D = [-1, -1/\kappa] \cup [1/\kappa, 1]$, and such that $|P(x)| \leq 1$ for all $x \in D$.*

A corollary of this claim is that if we can implement an $(\alpha, 0)$-block encoding of A, (that is, a block encoding of A/α), U, then there exists a $(2\kappa, \varepsilon)$-block encoding of A^{-1} that can be implemented using $O(\alpha\kappa \log(\kappa/\varepsilon))$ calls to U, using which we can approximately generate:

$$\frac{1}{2\kappa} A^{-1} |b\rangle |0\rangle + |\tilde{\psi}\rangle |1\rangle.$$

We can amplify the $|0\rangle$ part using

$$2\kappa \geq \left\| \frac{1}{2\kappa} A^{-1} |b\rangle \right\|^{-1}$$

[5]This complexity can be slightly improved in the log factors, but we omit the details. See, for example, [Gil19].

repetitions (via Theorem 2.2.11, amplitude amplification), since if we assume A has spectrum in D (which assumes $\|A\| \leq 1$), then A^{-1} has smallest eigenvalue 1. This gives a total cost of

$$O(\kappa^2 \alpha \log(\kappa/\varepsilon)).$$

For example, if A is given via sparse access (Definition 6.2.1), then we can get a block encoding of A with $\alpha = s$. The dependence on κ can be improved from quadratic to linear using more involved techniques [CGJ19].

Bibliography

[Aar15] Scott Aaronson. Read the fine print. *Nature Physics*, 11(4):291–293, 2015. 4

[AGJ20] Simon Apers, András Gilyén, and Stacey Jeffery. A unified framework of quantum walk search. In *Proceedings of the 38th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 6:1–6:13, 2020. arXiv: 1912.04233 13

[AGJK20] Andris Ambainis, András Gilyén, Stacey Jeffery, and Martins Kokainis. Quadratic speedup for finding marked vertices by quantum walks. In *Proceedings of the 52nd ACM Symposium on the Theory of Computing (STOC)*, page 412–424, 2020. arXiv: 1903.07493 12

[AS19] Simon Apers and Alain Sarlette. Quantum fast-forwarding: Markov chains and graph property testing. *Quantum Information and Computation*, 19(3&4):181–213, 2019. arXiv: 1804.02321 13

[BHMT02] Gilles Brassard, Peter Høyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. In *Quantum Computation and Quantum Information: A Millennium Volume*, volume 305 of *Contemporary Mathematics Series*, pages 53–74. AMS, 2002. arXiv: quant-ph/0005055

[CGJ19] Shantanav Chakraborty, András Gilyén, and Stacey Jeffery. The power of block-encoded matrix powers: improved regression techniques via faster Hamiltonian simulation. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 33:1–33:14, 2019. arXiv: 1804.01973 15

[Gil19] András Gilyén. *Quantum Singular Value Transformation & Its Algorithmic Applications*. PhD thesis, University of Amsterdam, 2019. 8, 10, 14

[Gre20] Tristan Greene. How quantum computers could make future humans immortal, 2020. https://thenextweb.com/news/how-quantum-computers-could-make-future-humans-immortal. 2

[GSLW19] András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. In *Proceedings of the 51st ACM Symposium on the Theory of Computing (STOC)*, pages 193–204, 2019. arXiv: 1806.01838 11, 14

[HHL09] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15):150502, 2009. arXiv: 0811.3171 3

[Kak24] Michio Kaku. Fighting cancer with quantum computing. https://www.noemamag.com/quantum-computing-could-make-cancer-more-like-the-common-cold/, 2024. 2

[KP17] Iordanis Kerenidis and Anupam Prakash. Quantum recommendation systems. In *Proceedings of the 8th Innovations in Theoretical Computer Science Conference (ITCS)*, pages 49:1–49:21, 2017. arXiv: 1603.08675 4, 7

[LC19] Guang Hao Low and Isaac L. Chuang. Hamiltonian simulation by qubitization. *Quantum*, 3:163, 2019. arXiv: `1610.06546` 14

[MNRS11] Frédéric Magniez, Ashwin Nayak, Jérémie Roland, and Miklos Santha. Search via quantum walk. *SIAM Journal on Computing*, 40(1):142–164, 2011. Earlier version in STOC'07. arXiv: `quant-ph/0608026` 6

[MRTC21] John M. Martyn, Zane M. Rossi, Andrew K. Tan, and Isaac L. Chuang. A grand unification of quantum algorithms. *Physical Review X*, 040203, 2021. arXiv: `2105.02859` 1, 14

[mtl] Too many to list. 2

[RWS$^+$17] Markus Reiher, Nathan Wiebe, Krysta M. Svore, Dave Wecker, and Matthias Troyer. Elucidating reaction mechanisms on quantum computers. *Proceedings of the National Academy of Sciences*, 114(29):7555–7560, 2017. 2

[Sze04] Mario Szegedy. Quantum speed-up of Markov chain based algorithms. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 32–41, 2004. arXiv: `quant-ph/0401053` 6

[Tan19] Ewin Tang. A quantum-inspired classical algorithm for recommendation systems. In *Proceedings of the 51st ACM Symposium on the Theory of Computing (STOC)*, pages 217–228, 2019. arXiv: `1807.04271` 4