Quantum Complexity Theory

Preliminaries

Hee Ryang Choi

SNU SQRT

October 13, 2022

Overview

Overview

Turing Machine

Turing machine can be defined as a 7-tuple $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$

Γ is a finite, non-empty set of tape alphabet symbols
$b \in \Gamma$ is the blank symbol
$\Sigma \subseteq \Gamma \setminus \{b\}$ is the set of input symbols
Q is a finite, non-empty set of states
$q_0 \in Q$ is the initial state
$F \subseteq Q$ is the set of final states
$\delta : (Q \setminus F) \times \Gamma \nrightarrow Q \times \Gamma \times \{L, R\}$ is the transition function

Church - Turing Thesis

Informal Description of Church - Turing Thesis

Every 'effectively calculable' function is a computable function

Definition of Computable Function

A partial function $f : \mathbf{N}^k \to \mathbf{N}$ is computable if and only if

- If $f(x)$ is defined, then the program will terminate on the input x with the value $f(x)$ stored in the computer memory.
- If $f(x)$ is undefined, then the program never terminates on the input x.

Computable function also called by recursive function or decidable function.

Computable Function

What function would be non-computable function?

> *Since Turing machines only map from non-negative to non-negative numbers, if any process in nature is found to map between different set of values, then that process cannot be run on a Turing machine*

Halting Problem

Halting Problem with no inputs

Show that given a Turing machine M, there is no algorithm to determine whether M halts when the input to the machine is a blank tape.

It can be solved by define a function $H(M) = \{$0 if machine doesn't halt with blank input, 1 if machine does halt with blank input$\}$.

Circuit

Definition of Circuit

A circuit is a triple (M, L, G), where M is a set of values, L is a set of gate labels(each of which is a function from M^i to M, for some non-negative integer i) and G is a labelled directed acyclic graph with L.

The labels of the leaves can also be variables which take values in M. So, if there are n leaves, the circuit regarded as a function from M^n to M.

Boolean Circuit

Boolean circuit is a finite directed acyclic graph over a basis B(like the set $\{AND, OR, NOT\}$), with n inputs and m outputs and following conditions.

- Each vertex corresponds to either a basis function or one of the inputs
- There is a set of exactly m nodes which are labeled as the outputs
- The edges must have ordering, to distinguish between different arguments to the same Boolean circuit

Basics of Universality

Universality of NAND

Show that the NAND gate can be used to simulate the AND, XOR and NOT gates, provided wires, ancillary bits and FANOUT; so NAND gates can implement any Boolean Circuits.

Circuit Family

A circuit family $\{C_n\}$ consists of circuits, indexed by a positive integer n. The circuit C_n has n input bits, any finite number of ancillary bits and output bits. Also, if we denote the length of output bit of C_n as $C_n(x)$, then if $m < n$ and x is at most m bits in length, then $C_m(x) = C_n(x)$.

Most important part of this definition is there is no restrictions on the circuit family, so it becomes possible to compute all sorts of functions. For example, halting function with restricted bits n, $h_n(x)$ is a function from n bits to 1 bits, so there MUST be some circuit C_n to calculate $h_n(x)$.

Uniform Circuit Family

We called circuit family is uniform when there is some algorithm running on Turing Machine which, upon input of n, generates a description; what gates are implemented in, how those gates are connected in, how many ancillary bits exists in, etc. in C_n.

Circuits vs Turing Machine

- Turing Machines are uniform models where the same computational device is used for all possible input lengths.
- Some Circuits may not be uniform models; in example, Boolean circuits are non-uniform models of computation with inputs of different lengths are processed by different circuits.

An individual computational problem is thus associated with a particular family of Boolean circuits. Also a uniformity condition is often imposed on these families, requiring the existence of some possibly resource-bounded Turing machine that, on input n, produces a description of the individual circuit.

Polynomial Time Uniform

A family of Boolean circuits $\{C_n : n \in \mathbf{N}\}$ is polynomial-time uniform if there exists a Deterministic Turing Machine, such that

- M runs in polynomial time
- For all $n \in \mathbf{N}$, M outputs a description of C_n on input 1^n

Some Terminologies

The **depth** of circuit is the length of the longest path from the input to the output, moving forward in time along wires.

The **size** of a circuit is the number of wires it has.

Motivation of Complexity Theory : How can we formally describe about Multi-Tape Turing Machine is trivially faster than Single-Tape Turing Machine

Strong Church - Turing Thesis

Any model of computation can be simulated on a Probabilistic Turing Machine with at most a polynomial increase in the number of elementary operations required.

Nondeterministic Turing Machine

Nondeterministic Turing Machine is same as Turing Machine, but only difference is that **transition function δ is not a function, just a relation** here

Probabilistic Turing Machine

Probabilistic Turing Machine has two tranistion function δ_1 and δ_2, **probabilistically select transition function** at each transition

Types of Problems

Decision Problem

Decision problem is a yes-or-no question on an infinite set of inputs. In the other words, a language $L \subseteq \{0, 1\}^*$ problem accept all inputs in L and reject all inputs not in L is decision problem

Search Problem

A problem is a search problem if there's an algorithmic way to verify the answer

Counting Problem

A problem C is a search problem if R is a search problem and for every x and y $C(x) = |\{y|R(x, y)\}|$ satisfied

Types of Problems

Promise problem is generalization of decision problem where the input is promise to belong to particular subset of all possible inputs; there may be inputs which are neither yes nor no.

Promise Problem

For language L_{YES} and L_{NO}, which must be disjoint, if problem accepts all inputs in L_{YES} and rejects all inputs in L_{NO}, then it called promise problem. $L_{YES} \cup L_{NO}$ is called the promise.

Terminologies

Alphabet

An alphabet Σ is a non-empty set of symbols, typically thought of as representing letters, characters, or digits

Words

Strings, also known as words, over an alphabet are defined as a sequence of the symbols from the alphabet set.

Formal Language

A formal language L over an alphabet Σ is a subset of Σ^*, that is, a set of words over that alphabet.

Computational Complexity

Complexity classes group computational problems by their resource requirements. To do this, computational problems are differentiated by upper bounds on the maximum amount of resources that the most efficient algorithm takes to solve them.

Time Bounds

The time complexity for an algorithm implemented with a Turing Machine M is defined as the function $t_M : \mathbf{N} \to \mathbf{N}$, where $t_M(n)$ is the maximum number of steps that M takes on any input of length n

Space Bounds

The space complexity of an algorithm implemented with a Turing Machine M is defined as the function $s_M : \mathbf{N} \to \mathbf{N}$, where $s_M(n)$ is the maximum number of cells that M uses on any input of length n

Computational Complexity

Polynomial-Time Computable

$p : \Sigma^* \rightarrow \Sigma^*$ is said to be a polynomial-time computable if there exists a polynomial-time deterministic Turing Machine that outputs $f(x)$ for every input $x \in \Sigma^*$

Polynomial-Bounded Function

$p : \mathbf{N} \rightarrow \mathbf{N}$ is said to be a polynomial-bounded function if and only if there exists a polynomial-time deterministic Turing Machine that outputs $1^{f(n)}$ on input 1^n for every $n \in \mathbf{N}$.

Basic Definitions

- $DTIME(t(n))$ is the set of all problems that are decided by an $O(t(n))$ time deterministic Turing Machine
- $NTIME(t(n))$ is the set of all problems that are decided by an $O(t(n))$ time nondeterministic Turing Machine
- $DSPACE(t(n))$ is the set of all problems that are decided by an $O(s(n))$ space deterministic Turing Machine
- $NSPACE(t(n))$ is the set of all problems that are decided by an $O(s(n))$ space nondeterministic Turing Machine

P and NP

P is the class of problems that are solvable by a deterministic Turing Machine in polynomial time. Formally it can be written as

$$P = \bigcup_{k \in \mathbf{N}} DTIME(n^k) \tag{1}$$

Or in another way; A language L is in P if and only if there exists a deterministic Turing Machine M, such that

- M runs for polynomial time on all inputs; said that P is polynomial-time computable
- For all x in L, M outputs 1
- For all x not in L, M outputs 0

Intuitively, we know that the complexity class P belongs to Promise Problem. In fact, Promise Problem is above most of the classes we think of.

P and NP

Maybe other definition of P from circuit family can be considered; A language L is in P if and only if there exists a polynomial-time uniform circuit family of Boolean circuits $\{C_n : n \in \mathbf{N}\}$, such that

- For all $n \in \mathbf{N}$, C_n takes n bits as input and outputs 1 bit
- For all x in L, $C_{|x|}(x) = 1$
- For all x not in L, $C_{|x|}(x) = 0$

There are many variations of P, such as Parity P, P, P-complete, PP, BPP, PH, IP, RP, ZPP, FP, etc. We will cover it later, if necessarily.

P and NP

NP is the class of problems that are solvable by a nondeterministic Turing Machine in polynomial time. Formally it can be written as

$$NP = \bigcup_{k \in \mathbf{N}} NTIME(n^k) \tag{2}$$

Or in another way; A language L is in *NP* if and only if there exists a deterministic Turing Machine M, polynomials p and q, such that

- For all x and y, M runs in time $p(|x|)$ on input (x, y)
- For all x in L, there exists a string y of length $q(|x|)$ such that $M(x, y) = 1$
- For all X not in L and all strings y of length $q(|x|)$, $M(x, y) = 0$

P and NP

Equivalence of two definitions of *NP* can be guaranteed by considering deterministic verifier. At first, suppose we have a deterministic verifier. Then

- A nondeterministic Turing Machine can simply run the verifier on all possible proof strings.
- If any proof is valid, some path will accept; if not, it will reject.

Conversely, suppose we have a nondeterministic Turing Machine M accepting a given language L. Then

- For L, M must find at least one accepting path. M gives description of this path to the verifier.
- Verifier deterministically simulate M, following supplied path. If verification accepts if every process is correct
- Otherwise, as there is no accepting input and M rejects input, verifier also rejects.

P and NP

Since every problems that deterministic Turing Machine can solve in polynomial time is trivially solved by nondeterministic Turing Machine in polynomial time, so $P \subseteq NP$. Here, determine the truthness of $P = NP$ is one of the Millennium Problems. $PSPACE$ is

formally defined as

$$PSPACE = \bigcup_{k \in \mathbf{N}} SPACE(n^k) \tag{3}$$

Because of Savitch's Theorem, $PSPACE$ is equivalent to $NPSPACE$, so we just use $SPACE(n^K)$ instead of $PSPACE(n^k)$.

P and NP

NP is contained in *PSPACE*, can be prooved by construct a PSPACE machine that loops over all proof strings and feeds each one to a polynomial-time verifier. It is possible because every polynomial time machine can only read polynomially may bits. In sense, it is trivial that *NP* is also contained in *EXPTIME*.

$$EXPTIME = \bigcup_{k \in \mathbf{N}} DTIME(2^{n^k}) \tag{4}$$

BPP and PP

BPP, bounded-error probabilistic polynomial time, is the class of decision problems solvable by a probabilistic Turing machine in polynomial time with an error probability bounded by 1/3 for all instances. A language L is in *BPP* if and only if there exists a probabilistic Turing Machine such that

- M runs for polynomial time on all inputs
- For all x in L, M outputs 1 with probability greater than or equal to 2/3
- For all x not in L, M outputs 1 with probability less than or equal to 1/3.

BPP and PP

BPP can be also defined by using deterministic Turing Machine; A language L is in *BPP* if and only if there exists a polynomial p and deterministic Turing machine M, such that

- M runs for polynomial time on all inputs
- For all x in L, the fraction of strings y of length $p(|x|)$ which satisfy $M(x, y) = 1$ is greater than or equal to 2/3
- For all x not in L, the fraction of strings y of length $p(|x|)$ which satisfy $M(x, y) = 1$ is less than or equal to 1/3

BPP and PP

A language *L* is in *PP* if and only if there exists a probabilistic Turing Machine such that

- *M* runs for polynomial time on all inputs
- For all *x* in *L*, *M* outputs 1 with probability greater than to 1/2
- For all *x* not in *L*, *M* outputs 1 with probability less than 1/2.

BPP and PP

PP can be also defined by using deterministic Turing Machine; A language L is in *PP* if and only if there exists a polynomial p and deterministic Turing machine M, such that

- M runs for polynomial time on all inputs
- For all x in L, the fraction of strings y of length $p(|x|)$ which satisfy $M(x, y) = 1$ is greater than 1/2
- For all x not in L, the fraction of strings y of length $p(|x|)$ which satisfy $M(x, y) = 1$ is less than 1/2

BPP and PP

What makes *PP* and *BPP* different?

It is simply because bound of *BPP* is not allowed to depend on the input, but *PP* is. For example, for *PP*, following can be considered.

- On a *YES* instance, output *YES* with probability $1/2 + 1/2^n$, where n is the length of the input
- On a *NO* instance, output *NO* with probability $1/2 - 1/2^n$, where n is the length of the input

So we cannot exceed Chernoff bound because of input-dependent-bound. This is why we can claim $BPP \subset PP$

Interactive Proof System

Interactive proof system is an abstract machine that models computation as the exchange of messages between two parties: a prover and a verifier. All interactive proof systems have two requirements:

- Completeness: if the statement is true, the honest prover can convince the honest verifier that it is indeed true.
- Soundness: if the statement is false, no prover, even if it doesn't follow the protocol, can convince the honest verifier that it is true, except with some small probability.

Interactive proof system can describes AM, IP, NP, MA, SZK, etc.

MA and AM

Informally, MA is a generalization of NP with verifier(Arthur) can use randomness. Formal definition is like this; A language L is in MA if there exists a polynomial time porbabilistic Turing Machine M and polynomials p, q such that for every input string x of length $n = |x|$,

- if x is in L, then
  $\exists z \in \{0,1\}^{q(n)} Pr_{y \in \{0,1\}^{p(n)}}(M(x,y,z) = 1) \geq 2/3$
- if x is not in L, then
  $\forall z \in \{0,1\}^{q(n)} Pr_{y \in \{0,1\}^{p(n)}}(M(x,y,z) = 0) \geq 2/3$

MA and AM

AM or $AM[2]$ is defined by adding information about Arthur's coin toss to Merlin. Formally, a language L is in AM if there exists a polynomial time porbabilistic Turing Machine M and polynomials p, q such that for every input string x of length $n = |x|$,

- if x is in L, then $Pr_{y \in \{0,1\}^{p(n)}}(\exists z \in \{0,1\}^{q(n)} M(x,y,z) = 1) \geq 2/3$
- if x is not in L, then $Pr_{y \in \{0,1\}^{p(n)}}(\forall z \in \{0,1\}^{q(n)} M(x,y,z) = 0) \geq 2/3$

MA and AM

$AM[k]$ is the set of problems that can be decided in polynomial time, with k queries and responses. For example, $AM[3]$ would start with one message from Merlin to Arthur, then a message from Arthur to Merlin and then finally a message from Merlin to Arthur. Here, the last message should always be from Merlin to Arthur.

Trivially, $AM[3]$ contains MA by ignoring last response from Merlin to Arthur. Also, it is known that for every $k \in \mathbf{N}$, $AM[k]$ equals to $AM[2]$. So AM contains MA.

Zero-Knowledge Proof

A zero-knowledge proof of some statement is kind of interactive proof system, with an additional condition:

- if the statement is true, no verifier learns anything other than the fact that the statement is true. In other words, just knowing the statement is sufficient to imagine a scenario showing that the prover knows the secret. This is formalized by showing that every verifier has some simulator that, given only the statement to be proved, can produce a transcript that ”looks like” an interaction between an honest prover and the verifier in question.

Zero-Knowledge Proof

Formally, an interactive proof system with (P, V) for a language L is a zero-knowledge proof system if for any probabilistic polynomial time verifier V, there exists a probabilistic polynomial time simulator S such that

$$\forall x \in L, z \in \{0,1\}^*, View_V\left[P(x) \leftrightarrow V(x,z)\right] = S(x,z) \tag{5}$$

where $View_V\left[P(x) \leftrightarrow V(x,z)\right]$ is a record of the interactions between $P(x)$ and $V(x,z)$, the prover P is modeled as having unlimited computation power.

SZK

A language L is in SZK if and only if it has a statistical zero-knoweldge interactive proof system.

Here, statistical zero-knowledge means that the distributions produced by the simulator and the proof protocol are not necessarily exactly the same, but they are statistically close, meaning that their statistical difference is a negligible function.

Oracle Machine

Oracle is a machine that provides a subroutine for solving instances of a chosen problem. For example, Turing Machine with halting problem oracle can solve halting problem.

The notation A^L means the complexity class of decision problems solvable by an algorithm in class A with an oracle for a language L.

Overview

Quantum Circuit

Quantum circuit is an acyclic network of quantum gates connected by wires. **Size** of quantum circuit and **depth** of quantum circuit is defined in same way as in classical circuit.

Unitary quantum circuit is a quantum circuit in which all of the gates correspond to unitary quantum operations. In this article, we use density matrix to describe quantum sytems.

Pure State and Qubits

A quantum physical system in a pure state is described by a unit vector in a Hilbert Space. We also denote a pure state in the Dirac Notation by $|\alpha\rangle$. The physical system which corresponds to quantum circuit consists of n quantum two-state particles, with a Hilbert Space $\mathbf{H}_2^n = \mathbf{C}^{\{0,1\}^n}$, which defined by tensor product of n Hilbert Spaces of one two-state particle.

We can choose a basis state of $\mathbf{H}_2^n$, which consists of the 2^n orthogonal states; $|i\rangle = |i_1\rangle |i_2\rangle \cdots |i_n\rangle$. Here, a pure state is generally a superposition of the basis states.

Mixed States

In general, a quantum system is not in a pure state. It is because of

- Knowledge about the system is partial
- System is not isolated from the universe

Mixed state is system in a situation above, or etc, can be denoted by mixture of pure states, $\{\alpha\} = \{p_k, |\alpha\rangle\}$, an ensemble of pure states. In this article, we will use density matrix notion to describe the states.

Density Matrix

Let's consider an ensemble $\{p_i, |\psi_i\rangle\}$. The density operator or the density matrix for the system is defined by the equation

$$\rho \equiv \sum_i p_i |\psi_i\rangle \langle\psi_i| \tag{6}$$

Suppose that the evolution of such closed quantum systems is described by the unitary operator U. Then the evolution of the density operator will be

$$\rho = \sum_i p_i |\psi_i\rangle \langle\psi_i| \xrightarrow{U} \sum_i p_i U |\psi_i\rangle \langle\psi_i| U^\dagger = U\rho U^\dagger \tag{7}$$

Measurement

Suppose we perform a measurement operator M_m, with initial state $|\psi_i\rangle$. Then the probability of getting result m is

$$p(m|i) = \langle\psi_i| M_m^\dagger M_m |\psi_i\rangle = tr(M_m^\dagger M_m |\psi_i\rangle \langle\psi_i|) \tag{8}$$

So the density operator ρ_m is

$$\rho_m = \sum_i p(i|m) |\psi_i^m\rangle \langle\psi_i^m| = \sum_i p(i|m) \frac{M_m |\psi_i\rangle \langle\psi_i| M_m^\dagger}{\langle\psi_i| M_m^\dagger M_m |\psi_i\rangle} \tag{9}$$

Characterization of Density Operators

An operator ρ is the density operator associated to some ensemble $\{p_i, |\psi_i\rangle\}$ if and only if it satisfies the conditions:

- Trace Condition : ρ has trace equal to one
- Positivity Condition : ρ is a positive operator

$$\begin{aligned} \langle\phi|\,\rho\,|\phi\rangle &= \sum_i p_i \langle\phi|\,|\psi_i\rangle\,\langle\psi_i|\,|\phi\rangle && (10)\\ &= \sum_i p_i |\,\langle\phi|\,|\psi_i\rangle\,|^2 && (11)\\ &\geq 0 && (12) \end{aligned}$$

Unitary Freedom in the Ensemble

Unitary Ensemble Theorem

The sets $|\tilde{\psi}_i\rangle$ and $|\tilde{\phi}_j\rangle$ generate the sane density matrix if and only if

$$|\tilde{\psi}_i\rangle = \sum_j u_{ij} |\tilde{\phi}_j\rangle \tag{13}$$

where u_{ij} is a unitary matrix of complex numbers, with indicies i and j. Also, if size of two set of vectors are different, we will ”pad” it by adding zero-vectors.

Because of two set of vectors generate the same density matrix, we can also gain a relationship between probability distributions of each set.

Unitary Freedom in the Ensemble

Suppose $|\tilde{\psi}_i\rangle = \sum_j u_{ij} |\tilde{\phi}_j\rangle$ for some unitary u_{ij}. Then

$$\sum_i |\tilde{\psi}_i\rangle \langle\tilde{\psi}_i| = \sum_{ijk} u_{ij} u_{ik}^* |\tilde{\phi}_j\rangle \langle\tilde{\phi}_k| \tag{14}$$

$$= \sum_{jk} \left(\sum_i u_{ki}^\dagger u_{ij} \right) |\tilde{\phi}_j\rangle \langle\tilde{\phi}_k| \tag{15}$$

$$= \sum_{ik} \delta_{kj} |\tilde{\phi}_j\rangle \langle\tilde{\phi}_k| \tag{16}$$

$$= \sum_j |\tilde{\phi}_j\rangle \langle\tilde{\phi}_j| \tag{17}$$

Unitary Freedom in the Ensemble

Conversely, suppose

$$A = \sum_i |\tilde{\psi}_i\rangle \langle\tilde{\psi}_i| = \sum_j |\tilde{\phi}_j\rangle \langle\tilde{\phi}_j| \tag{18}$$

Let $A = \sum_k \lambda_k |k\rangle \langle k|$ be a decomposition of A such that the states $|k\rangle$ are orthonormal, and the λ_k are strictly positive. Let $|psi\rangle$ be any vector orthonormal to the space spanned by the $|\tilde{k}\rangle$, so $\langle\psi| |\tilde{k}\rangle \langle\tilde{k}| |\psi\rangle = 0$ for all k, thus we see that

$$0 = \langle\psi| A |\psi\rangle = \sum_i \langle\psi| |\tilde{\psi}_i\rangle \langle\tilde{\psi}_i| |\psi\rangle = \sum_i | \langle\psi| |\tilde{\psi}_i\rangle |^2. \tag{19}$$

Unitary Freedom in the Ensemble

Thus $\langle\psi| |\tilde{\psi}_i\rangle = 0$ for all i and all $|\psi\rangle$ orthonormal to the space spanned by the $|k\rangle$. It follows that each $|\psi\rangle$ can be expressed as a linear combination of the $|\tilde{k}\rangle$, $|\tilde{\psi}_i\rangle = \sum_k c_{ik} |\tilde{k}\rangle$. Since $A = \sum_k |\tilde{k}\rangle \langle\tilde{k}| = \sum_i |\tilde{\psi}\rangle \langle\tilde{\psi}|$ we see that

$$\sum_k |\tilde{k}\rangle \langle\tilde{k}| = \sum_k l \left(\sum_i c_{ik} c_{il}^*\right) |\tilde{k}\rangle \langle\tilde{l}| . \tag{20}$$

The operators $|\tilde{k}\rangle \langle\tilde{l}|$ are linearly independent, so we can append extra columns to c, without the loss of the unitarity.

Reduced Density Operator

Suppose we have physical systems A and B, whose state is decribed by a density operator ρ^{AB}. The reduced density operator for system A is defined by

Reduced Density Operator

$$\rho^{A} \equiv tr_{B}(\rho^{AB}), \tag{21}$$

where tr_B is a map of operators known as partial trace over system B. The partial trace is defined by

Partial Trace

$$tr_{B}(|a_1\rangle\langle a_2| \otimes |b_1\rangle\langle b_2|) \equiv |a_1\rangle\langle a_2|\, tr(|b_1\rangle\langle b_2|) \tag{22}$$

Schmidt Decomposition

Schmidt Decomposition

Suppose $|\psi\rangle$ is a pure state of a composite system, AB. Then there exist orthonormal states $|i_A\rangle$ for system A, and orthonormal states $|i_B\rangle$ of system B such that

$$|\psi\rangle = \sum_i \lambda_i |i_A\rangle |i_B\rangle \tag{23}$$

where λ_i are non-negative real numbers satisfying $\sum_i \lambda_i^2 = 1$ known as Schmidt coefficients.

Purification

Purification Theorem

Suppose we are given a state ρ^A of a quantum system A. It is possible to introduce another system, which denoted by R, and define a pure state $|AR\rangle$ for the joint system AR such that $\rho^A = tr_R(|AR\rangle\langle AR|)$.

To prove this, suppose ρ^A has orthonormal decomposition $\rho^A = \sum_i p_i |i^A\rangle\langle i^A|$. Now consider a system R which has the same state space as A, with orthonoraml basis $|i^R\rangle$.

Unitary Freedom in the Ensemble

Define a pure state for the combined system

$$|AR\rangle \equiv = \sum_i \sqrt{p_i} |i^A\rangle |i^R\rangle . \tag{24}$$

Now calculate the reduced density operator for A corresponding to the AR

$$\begin{aligned} tr_R(|AR\rangle \langle AR|) &= \sum_{ij} \sqrt{p_i p_j} \langle i^A| |j^A\rangle tr(\langle i^R| |j^R\rangle) &(25)\\ &= \sum_{ij} \sqrt{p_i p_j} \langle i^A| |j^A\rangle \delta_{ij} &(26)\\ &= \sum_{ij} p_i \langle i^A| |i^A\rangle &(27)\\ &= \rho^A &(28) \end{aligned}$$

Universality Theorem

Universality Theorem

Unitary transformation U on n qubits which take $\Omega(2^n log(1/\epsilon)/log(n))$ operations to approximate by a quantum circuit implementing an operation V such that $E(U, V) \leq \epsilon$.

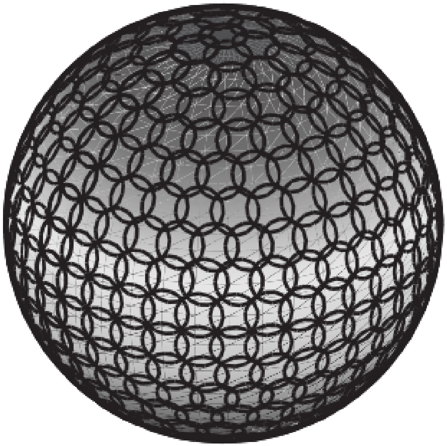
Unitary Operator Error

Error when V is implemented instead of U is

$$E(U, V) \equiv max_{|\psi\rangle|(U-V)|\psi\rangle|}.(29)$$

where the maximum is over all normalized quantum states $|\psi\rangle$ in the state space.

Universality Theorem



Universality Theorem

Where surface area of a *k*-sphere of radius *r* is

$$S_k(r) = \frac{2\pi^{\frac{k+1}{2}} r^k}{\Gamma(\frac{k+1}{2})} \tag{30}$$

and volume of a *k*-sphere of radius *r* is

$$V_k(r) = \frac{2\pi^{\frac{k+1}{2}} r^{k+1}}{(k+1)\Gamma(\frac{k+1}{2})}. \tag{31}$$

So number of patches we need to cover the state space will be like

$$\frac{\sqrt{pi}\Gamma(2^n - \frac{1}{2})(2^{n+1} - 1)}{\Gamma(2^n)\epsilon^{2^{n+1}-1}} \tag{32}$$

Universality Theorem

Suppose that objective quantum circuit has f input qubits, g gates, and we have m unitary transformation on n qubits. Then

$$O(n^f gm) \geq \Omega(\frac{1}{\epsilon^{2^{n+1}-1}}) \tag{33}$$

Universality Theorem

Solovay-Kitaev Theorem

Let **G** be a finite set of elements in $SU(2)$ containing its own inverses, such that $\langle \mathbf{G} \rangle$ is dense in $SU(2)$. Let $\epsilon > 0$ be given. Then $\mathbf{G}_l$ is an ϵ-net in $SU(2)$ for $l = O(log^c(1/\epsilon)$, for c nearly in 4.

Universality Theorem 2

Arbitrary unitary operation U on n qubits may be approximated to within distance ϵ using $O(n^2 4^n log^c(n^2 4^n/\epsilon))$ gates.