Complexity Theory

How difficult is it to solve to estimate the ground state energy of a local Hamiltonian?

A glib answer to this question is that the difficulty depends on the specific Hamiltonian!

For some Hamiltonians, we can immediately guess the ground state e.g. $H = -\sum_{i=1}^{n} Z_i Z_{i+1}$

For other classes of Hamiltonians, simplifying assumptions may allow computational physicists and theoretical computer scientists to develop **efficient classical simulation algorithms**.

However, the exponential time hypothesis says that we expect 3-SAT (which is a local Hamiltonian problem) to take exponential time to solve on a classical computer. Similarly, we expect that for a general local Hamiltonian it takes an exponential time to find the ground state energy, even using a quantum computer.

This statement about the **worst-case** complexity for a class of local Hamiltonians also holds if we restrict the Hamiltonian to be 2-local on a translation-invariant 2D square lattice.

Classical Complexity Theory

Quantum complexity theory proceeds in a way that is strongly (and beautifully) analogous to classical complexity theory. To understand the former we first need to learn the latter.

The starting point is to abstractly characterize efficient classical deterministic computation.

Informally, the complexity class P consists of all problems that can be solved in a time that scales polynomially with the size of the input description of the problem. Theoretical computer scientists regard P as the class of problems that are easy to solve on a classical computer.

The class P includes many of the provably efficient algorithms studied in CS courses e.g. sorting a list, multiplying integers, finding the shortest path between two points of a graph, and so on.

A crucial point is that a problem is in P even if the algorithm runs for time n^{100} on inputs of length n. Or it could run for time $10^{10^{10^{10}}} n$. In what sense is this efficient?

Classical Complexity Theory

More formally, P is a set of languages. A Language is a set of bit strings. A language L is in P if polynomial-time deterministic classical circuits can *recognize* L, in the following sense.

Let Σ be a finite alphabet, and let Σ^* denote the (infinite) set of all words (of arbitrary length) formed from the alphabet Σ. A subset L of Σ^* is called a language. (this * notation is called the Kleene Star)

It is important that L is infinite and contains strings of varying size. For example, the set of all 3-SAT instances with a satisfying assignment is a language L. So is the set of all R-sided polygons with area A, and so on. Languages are extremely general.

To recognize strings of varying length, we will also need circuits of varying size. A circuit verifier (or just a verifier) is a family of circuits $\{V_r\}_{r\in\mathbb{N}}$, such that each V_r has r gates (chosen from some set of combinatorically local gates like AND, OR , NOT) , and on each input x, V_r outputs a single bit, $V_r(x)$.

The bit output by the verifier signifies the answer to a YES/NO question, or we say it either “accepts” (outputs 1) or “rejects” (outputs 0).

Classical Complexity Theory

A Language L is in P if there exists a polynomial p and a verifier $\{V_r\}_{r\in\mathbb{N}}$ such that for all $x \in L$

$$V_{p(|x|)}(x) = 1$$

(here |x| is the size of the bit string x), and for all $x \notin L$,

$$V_{p(|x|)}(x) = 0$$

Semi-formally, $L \in P$ if there is a polynomial-time verifier that distinguishes whether $x \in L$ or $x \notin L$

The main purpose of the formality is to properly handle infinite sets. Infinite sets make mathematics richer (and incomplete). Infinity is the reason that “P vs NP” is a mathematically deep question.

Even though computations are always finite sized, we can consider infinite families of computations which together have unbounded size.

Classical Complexity Theory

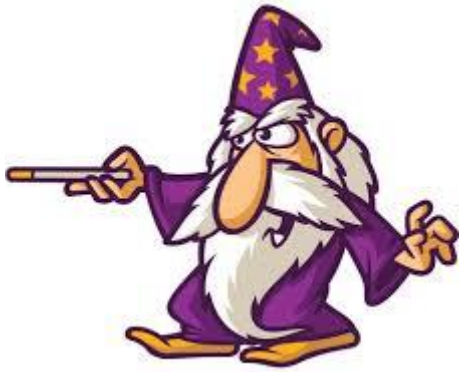
Now we turn to the class NP. The name refers to a “nondeterministic verifier.”

Very informally, NP is the class of problems you can solve efficiently if you are infinitely lucky.

If you’re infinitely lucky, you can decide whether a 3-SAT instance is satisfiable. Just guess the satisfying assignment, and if you’re right everyone can see it, and if not then there must be no satisfying assignment.

Similarly, you’re so lucky that you can factor integers. Just guess the prime factorization, and then it is easy for anyone to check.

Instead of luck, you can think of Merlin, the all-powerful wizard, who hands you a string to help you see the answer to the problem.



Classical Complexity Theory

Semi-formally, a problem is in NP if there exists a polynomial sized **witness** (a bit string) which allows you to solve the problem in polynomial time.

A Language L is in NP if there exists polynomials p and q and a verifier $\{V_r\}_{r\in\mathbb{N}}$ such that for each $x \in L$ there exists a bit string y with $|y| = q(|x|)$ such that

$$V_{p(|x|)}(x, y) = 1$$

And for all $x \notin L$, for all strings y of size $|y| = q(|x|)$ we have

$$V_{p(|x|)}(x, y) = 0$$

The string y is essentially the answer to the problem described by x. If the answer is YES (i.e. $x \notin L$) then (1) there is a succinct classical bit string y that acts as witness for x being in L, and (2) a classical computer can use the witness y efficiently to check that x is in L.

If the answer is NO (i.e. $x \notin L$) then there is no bit string *y* that could fool the verifier into accepting. The verifier always rejects regardless of the witness.

Classical Complexity Theory

The fact that the verifier always rejects NO instances is crucial to making NP an interesting definition.

Consider a 3-SAT problem. If it has a satisfying assignment, Merlin can give that assignment to me and I can check it in polynomial time by "plug and chug." But if there is no satisfying assignment, then there is nothing he could give me that would lead me to falsely conclude the formula is satisfiable. I'll find a violated clause.

Similarly, I may ask "does the integer N have a factor less than R?". If the answer is yes then Merlin can give me the factor and I can check it with division. If the answer is no, then no integer he gives me will lead me to falsely conclude that the answer is yes, since I can do division efficiently and see the remainder is non-zero.

Some of the problems in NP, like 3-SAT, are believed not to be in P. This is unsurprising since we neither have infinite luck nor all-powerful wizard assistants.

But NP is not all powerful. Suppose I have a general local Hamiltonian, and I ask whether its ground state energy is nonzero. What witness would Merlin try to send? How would I check it?

Classical Complexity Theory

From the definition, NP can recognize all the languages in P. Therefore $P \subseteq NP$. It is strongly believed that

$$P \subset NP$$

But this remains one of the greatest unsolved problems in modern mathematics. The Clay Mathematics institute offers a million dollars for a proof that $P \subset NP$ and hence $P \neq NP$. Note that they do not consider claimed proofs that $P = NP$.

Since NP contains lots of languages that are easy to recognize, like the ones in P, we would like some way to talk about the hardest problems in NP.

To build towards doing this, we introduce a notion of reducing one problem to another.

We say there is a polynomial-time reduction from problem A to problem B if there is a polynomial-time classical algorithm that transforms all the inputs for A into inputs for B. Therefore if $B \in NP$, and A is poly-time reducible to B, then $A \in NP$.

$$(L \in NP) \wedge (\text{L' is poly-time reducible to L}) \implies L' \in NP$$

Classical Complexity Theory

A Language L is called **NP-hard** if every Language L' in NP is poly-time reducible to L.

Informally, if a problem is NP-hard then it is "at least as hard (difficult) as all of the problems in NP."

The complexity class NEXP is defined analogously to NP, but allows for exponentially large witnesses and verifiers. We can prove $NP \subset NEXP$ (using the time hierarchy theorem), so NEXP is a strictly more powerful class. The point is that many problems in NEXP are NP-hard, but not representative of problems in NP.

Therefore we are most interested in the NP-hard problems that can also be solved in NP, and these languages are called **NP-complete**. This is the interesting part of NP.

From these definitions alone, it is not obvious that any NP-complete languages exist.

The cornerstone of NP-completeness is the **Cook-Levin theorem**, which states that **3-SAT is NP-complete**. After we prove this we can find an abundance of NP-complete languages by reducing them to 3-SAT.

Classical Complexity Theory

Let $L \in NP$. The Cook-Levin theorem is based on a poly-time reduction from L to 3-SAT. Each $x \in L$ will be mapped to a satisfiable 3-SAT instance with at most $poly(|x|)$ variables and clauses. Each $x \notin L$ is mapped to an unsatisfiable 3-SAT instance with at most $poly(|x|)$ variables and clauses.

Fix x , and consider the verifier circuit V of the appropriate size. The input for V is the string x, along with a witness string y with size $poly(|x|)$. If $x \in L$ we know there is such a y that makes the verifier accept, and if $x \notin L$ then the verifier will reject for every possible witness y.

The verifier circuit V can be taken to consist of classical logic gates with at most 2 inputs and one output. Examples of such universal classical gate sets include {AND, OR, NOT}, or just NAND by itself.

For each gate in the circuit, we can write down a Boolean function on the inputs and outputs that is TRUE if and only if the inputs match the outputs.

$$(i_1 \wedge i_2) \Leftrightarrow o_1$$

Classical Complexity Theory

Therefore logic gates can be enforced by Boolean clauses. The Cook-Levin proof is based on using clauses to enforce the history of a valid classical computation.

Suppose the state of a circuit at time t the bit string x_t , so that $x_{t,i}$ is the state of the i-th bit at time t.

If two bits $x_{t,i}, x_{t,j}$ pass through a gate at time t, then the output bit $x_{t+1,k}$ is determined by the input bits, and there is a corresponding Boolean proposition

$$R(x_{t,i}, x_{t,j}, x_{t+1,k})$$

That is TRUE if and only if the inputs match the outputs. The entire history of the circuit can be enforced by the conjunction (AND) of many such clauses.

Reducing the 3-local propositions to CNF is an exercise. Or one can define a new 3-CSP called CIRCUIT-SAT which we are showing to be NP-complete, and reduce CIRCUIT-SAT to 3-SAT later on.

The reason k-SAT is NP-complete only for $k \geq 3$ is because universal circuits need gates with at least two inputs and one output.

Classical Complexity Theory

The clauses enforce the correct operation of all the gates in the circuit, and also enforce the initial state to be x. But what about the part of the input that corresponds to the witness y? This is unconstrained.

The deterministic part of the circuit is just a proposition: start from this state, go to this state, then this state, and so on until you output the bit 1.

$$R(x_1 = x, x_2, ..., x_{T,out} = 1)$$

But the nondeterministic part of the circuit is the witness y: the answer to the problem you can check efficiently, the lucky guess, the wizard's gift. Therefore we ask whether there exists some y that can make the overall proposition true ("take x,y as input, operate the circuit correctly, output 1")

$$(\exists y) R(x_1 = (x, y), x_2, ..., x_{T,out} = 1)$$

This quantifier, followed by R expressed in CNF, is what makes this a SAT problem. The search for an acceptable witness is the same as the search for a satisfying assignment. This is the Cook-Levin theorem.

Classical Complexity Theory

If the input x has size |x| and runs for time T = poly(|x|), then our reduction to 3-SAT uses an additional $|x| \cdot T$ variables to represent the time evolution of the initial state through the circuit. This is an example of how polynomial blowups naturally arise from reductions.

Solving 3-SAT is as difficult as solving any problem in NP. But any 3-SAT instance can be expressed as a local Hamiltonian problem: asking whether a particular Hamiltonian (which happens to be diagonal in the Pauli Z basis) has a ground state energy of 0 or not.

Finding quantum ground states is at least as hard, in the worst-case, as performing nondeterministic computation.

Once we know 3-SAT is NP-complete, we can find other NP-complete problems by reducing them to 3-SAT. The theory of NP-completeness arose in the late 60s/early 70s as computer scientists realized the connection between dozens of difficult combinatorial problems they had studied, which did not appear to have efficient algorithms but had concise efficiently checkable proofs. Karp, “Reducibility Among Combinatorial Problems”, 1972.

Classical Complexity Theory

SUBSET SUM is NP-complete: given a set S of n integers, decide whether there is a subset of S whose elements sum to zero.

K-COLORING is NP-complete: a k-coloring of a graph with n vertices assigns one of k colors to each vertex of the graph, in such a way that no two vertices sharing an edge have the same color. 3-COLORING is NP-complete.

MAXCUT is NP-complete. The max cut of a graph is the subset of vertices that maximizes the number of edges leaving the set. The decision version ("is there a subset that cuts more than M edges?") is NP-complete.

FACTORING is an example of a problem that is in NP, believed not to be in P, but also not believed to be NP-complete. Another such problem is GRAPH ISOMORPHISM (deciding whether two graphs are isomorphic).

P also has complete problems, which tend to look like things that are straightforward and laborious. Evaluating a Boolean proposition when all the variables are literals is P-complete.

As humans, we try to solve special cases of NP problems by using creativity or luck to get the witness y. If P = NP this creativity would be unnecessary: we could proof all provable theorems by plugging and chugging for poly time. (all mathematical theorems with efficiently readable proofs are in NP)

Classical Complexity Theory

A related complexity class is co-NP. A problem is in co-NP if its negation is in NP. In other words, there is an efficient witness for NO instances.

3-SAT is not believed to be in co-NP. If it were, we should be able to give an efficient witness that shows the formula is not satisfiable. Therefore it seems clear that

TAUTOLOGY is NP-complete. Given a proposition, decide whether it is a tautology (true for all assignments of the literals). If the answer is NO then it is easy to give an efficient witness. Note the rule for negating quantifiers:

$$\neg(\exists y)R(x,y) = (\forall y)\neg R(x,y)$$

If an NP-complete problem is in co-NP, this would imply NP = co-NP which is not expected for the reason above.

FACTORING is in $\mathrm{NP} \cap \text{co-NP}$. Does the number N have a factor less than M? If the answer is yes, the witness is the factor. If the answer is no, the witness is a list of all the prime factors. This is one of the main reasons we believe factoring is not NP-complete.

Classical Complexity Theory

On the way from deterministic classical computation and quantum computation, we first want to treat probabilistic classical computation.

The set of problems that can be efficiently solved by a classical computer that can flip random coins is called BPP ("bounded-error probabilistic polynomial time"). It's just a poly-time circuit that can flip coins.

The bounded-error stipulation is the following. If the answer is YES the circuit outputs 1 with probability 2/3, and if the output is NO it outputs 1 with probability at most 1/3.

The probability to output 1 in the yes instance is called completeness, c = 2/3. The probability to output 1 in the NO instance is the soundness, s = 1/3. The reason we choose (c,s) to be these constants is that their specific values hardly matter. The reason is that we can repeat the circuit many times ("parallel repetition") to increase the completeness exponentially towards 1, and decrease the soundness exponentially near zero.

$$BPP(c,s) = BPP(1-(1-c)^L, s^L) \quad , \quad L = poly(n)$$

Using parallel repetition, we just need c and s to be constants bounded away from ½. Without this stipulation we can have probabilities exponentially close to ½, and that model of computation is too powerful to be realistic.

Classical Complexity Theory

The class BPP represents the set of problems that can be solved by efficient randomized classical algorithms, and hence it is the true extent of what we consider to be feasible with classical computation.

We believe that P = BPP because pseudo-random number generators are seemingly capable of replacing true randomness. But we are as far from proving P = BPP as we are from proving P != NP.

For technical reasons, BPP does not have any complete problems. Neither do any of the classes involving bounded error, and it is because of the requirement to reject all $x \notin L$. There are some x that can output 1 with a probability outside of the bounded range.

To solve this problem, which is just a technicality but appears everywhere, we define promise problem/languages,

$$L = L_{\mathrm{YES}} \cup L_{\mathrm{NO}} \quad , \quad L_{\mathrm{YES}} \cap L_{\mathrm{NO}} = \emptyset$$

Where the BPP machine only needs to decide whether $x \in L_{\mathrm{YES}}$ or $x \in L_{\mathrm{NO}}$, promised that one of these two possibilities is the case. We don't care what happens when x is outside of $L_{YES} \cup L_{NO}$.

Classical Complexity Theory

The class BPP represents the set of problems that can be solved by efficient randomized classical algorithms, and hence it is the true extent of what we consider to be feasible with classical computation.

We believe that P = BPP because pseudo-random number generators are seemingly capable of replacing true randomness. But we are as far from proving P = BPP as we are from proving P != NP.

For technical reasons, BPP does not have any complete problems. Neither do any of the classes involving bounded error, and it is because of the requirement to reject all $x \notin L$. There are some x that can output 1 with a probability outside of the bounded range.

To solve this problem, which is just a technicality but appears everywhere, we define promise problem/languages,

$$L = L_{\mathrm{YES}} \cup L_{\mathrm{NO}} \quad , \quad L_{\mathrm{YES}} \cap L_{\mathrm{NO}} = \emptyset$$

Where the BPP machine only needs to decide whether $x \in L_{\mathrm{YES}}$ or $x \in L_{\mathrm{NO}}$, promised that one of these two possibilities is the case. We don't care what happens when x is outside of $L_{YES} \cup L_{NO}$.

Classical Complexity Theory

We can also consider a version of NP where the verifier is allowed to be a BPP machine. This class is called MA, which is short for Merlin Arthur. The all-powerful Merlin gives the witness string y to the human King Arthur, who has a BPP machine, and for YES instances Arthur outputs 1 with completeness probability at least 2/3, and for NO instances he outputs 1 with soundness probability at most 1/3.

The class MA was defined by Babai in 1985, but the first Promise MA-complete problem was given in 2006 by Bravyi and Terhal as a restricted kind of local Hamiltonian problem. "Merlin-Arthur Games and Stoquastic Complexity", 2006.

If P = BPP, then MA = NP. But we feel far from proving this. There is also another class AM, Arthur Merlin, where Arthur first flips some coins, and Merlin sends a witness that depends on the coins, and then Arthur verifies the witness. $MA \subseteq AM$, but they don't seem to be equal. An interesting problem in AM is estimating the volume of an exponentially large subset of bit strings.

Classical Complexity Theory

Moving up the ladder a bit, we saw that NP problems have the form

$$(\exists y)R(x,y)$$

And co-NP problems have the form

$$(\forall y)R(x,y)$$

Therefore if we could solve both classes of problems if we could ascertain the truth value of formulas like

$$(\exists z)(\forall y)R(x,y,z)$$

This complexity class is called "the second level of the polynomial hierarchy". We can keep climbing by alternating quantifiers again to get to the 3rd level,

$$(\forall w)(\exists z)(\forall y)R(x,y,z,w)$$

The union of all these classes (with a number of quantifiers independent of the input length, but including unbounded numbers of them) is called the polynomial hierarchy, PH. It is recursively defined, and the complete problems tend to be recursive. CIRCUIT MINIMIZATION is complete for the 2nd level.

Classical Complexity Theory

An example of an overpowered probabilistic class if PP, which is defined analogously to BPP but with no bound on the error. In a YES instance the probability of outputting 1 is greater than ½, and in a NO instance the probability of outputting 1 is less than ½. Because the circuit only has polynomial size, the probabilities can only get as small as $2^{-poly(n)}$, so

$$p_{\mathrm{YES}} = \frac{1}{2} + 2^{-poly(n)} \quad , \quad p_{\mathrm{NO}} = \frac{1}{2} - 2^{-poly(n)}$$

An example of a PP-complete problem is MAJSAT. Given a Boolean proposition, decide whether the majority of possible assignments satisfy the proposition or not. This is much tougher than 3-SAT!

The proof that MAJSAT is in PP is simple. Choose a random assignment, if it evaluates to true then output YES (claim that the majority of assignments are satisfiable). If that random assignment evaluates to false, output NO. Clearly this is a wild guess based on extremely limited information. But PP only demands we do slightly better than a coin toss. The proof that MAJSAT is PP-hard is not much more difficult.

In 1989 Toda proved that P^{PP}, the class of problems that can be solved in polynomial time with access to a PP oracle (a machine that instantly solves PP problems) contains the entire PH.

Classical Complexity Theory

Another powerful class is #P, which counts the number of accepting inputs to a nondeterministic verifier (the number of witness y which will cause the verifier to accept). The natural complete problem is #SAT, counting the number of satisfying assignments to a Boolean proposition.

Many problems related to counting exponentially large sets are #P complete, including physics problems related to the exact computation of partition functions.

#P and PP have comparable power because $P^{\#P} = P^{PP}$, but note the different definitions of #P and PP make them suited to capturing different kinds of problems.

Classical Complexity Theory

The final complexity class of our whirlwind tour is PSPACE, the set of problems that can be solved using polynomial space and unlimited time. Does PSPACE contain PP?

Classical Complexity Theory

The final complexity class of our whirlwind tour is PSPACE, the set of problems that can be solved using polynomial space and unlimited time. Does PSPACE contain PP? Yes, because MAJSAT is in PSPACE.

An example of a PSPACE-complete problem is OEOTL (OTHER END OF THE LINE). Suppose we have a graph with vertices labeled by n bit strings. Every vertex has degree at most 2. For any vertex v we can see the strings of vertices that are neighbors of v. Given that 00…0 has only one neighbor, find the other end of the line that starts with 00…0.

Another complete problem for PSPACE is QSAT, deciding the truth value of formulas with poly(|x|) many alternating quantifiers,

$$(\forall y_1)(\exists y_2)...(\forall y_{k-1})(\exists y_k)R(x, y_1, ..., y_k)$$

Which gives the sense that PSPACE is more powerful than PH. PSPACE is contained in EXPTIME. (despite having unlimited time, there is no point in taking longer than exponential time when you only have polynomial space to work with).

Classical Complexity Theory