What is (Computational) Complexity Theory?

Complexity theory and quantum computing

Computational complexity theory focuses on classifying **computational problems** according to their *resource usage*, and relating classes of problems to each other. (Wikipedia)

1. Examples of computational problems

1. **Multiplication**. Given a pair of integers (m, n) as input, compute their product

2. **Factoring**. Given a composite number n as input, decompose n into a product of smaller integers

3. **Satisfaction** (SAT). Given a set of constraints on a set of Boolean variables, decide if the constraint system is satisfiable or not

   $(x_1 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) \wedge \cdots$

   $\bigwedge_{j=1}^{m} C_j$

$\vee$ Input	Output	$\wedge$ Input	Output	$\neg$ Input	Output
00	0	00	0	0	1
01	1	01	0	1	0
10	1	10	0		
11	1	11	1		

$x_1 = 1, \ x_2 = 0, \ \cdots$

4. **Counting** (#SAT). Given a set of constraints on a set of Boolean variables, compute the number of different solutions to the constraint system

5. **True Quantified Boolean Formula**. Given a formula in quantified propositional logic where every variable is quantified by either existential or universal quantifiers at the beginning of the formula, decide if the formula evaluates to true

   $\forall x_1 \exists x_2 \forall x_3 \forall x_4 ((x_1 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) \wedge \cdots)$

Remarks

1. A *family* of problem instances, indexed by the input size

2. Decision problems, search problems, promise problems

- Decision problem: a set $A \subseteq \Sigma^*$ where $\Sigma = \{0,1\}$

- Promise problem: disjoint sets $A_{yes}, A_{no} \subseteq \Sigma^*$

problem.cnf
p 4 2
1 3 -4 0
1 -2 -3 4 0

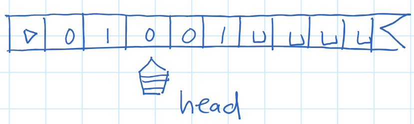
size $O(m \log n)$

2. Computational resources

Time, space, depth, …

Computational models

- *Turing machines*



A brief discussion of TMs

Γ: tape alphabet ← finite

Q: $q_0, q_1, \cdots, q_{acc}$ state set

Transition rule $\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$

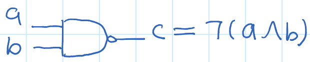
Configuration of a TM

$C_0 \to C_1 \to C_2 \to C_3 \to \cdots$

a. Current state q

b. Current tape content

c. Current head position



- Classical and quantum *circuit models*

A circuit family $\{C_m\}$ solves a problem A if for all $x \in A_{yes}$ $C_{|x|}(x) = 1$ and for all $x \in A_{no}$ $C_{|x|}(x) = 0$.

Classical circuits are not necessarily reversible

Uniform circuit family

Efficient computation

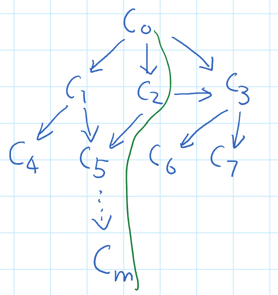
Polynomial-time (mathematically simple, model-independent)

3. P vs NP

- P: A promise problem A is in P if and only if there exists a polynomial-time *deterministic* Turing machine that accepts every string $x \in A_{yes}$ and rejects every string $x \in A_{no}$

- NP: *Non-deterministic* polynomial-time

Multiple choices

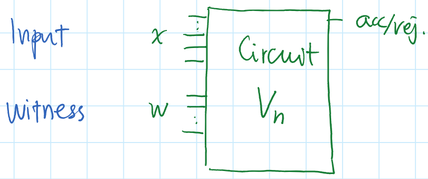


Non-deterministic transition rule $\delta: Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$

A non-deterministic Turing machine accepts if there exists an accepting computational path

Configurations of a non-deterministic TM

A proof-verification perspective of NP



A problem A is in NP if there exist a polynomial p and a polynomial-time uniform family of circuits $\{V_n\}$ acting on $n + p(n)$ input bits such that

1. If $x \in A_{yes}$, there is a $w \in \Sigma^{p(n)}$ s.t. $V_n(x, w) = 1$.
2. If $x \in A_{no}$, for all $w \in \Sigma^{p(n)}$ s.t. $V_n(x, w) = 0$.

SAT is NP-complete (1) SAT $\in$ NP (2) all problems in NP reduces to SAT

4. Church-Turing thesis

A problem is (*efficiently*) computable by an **effective method** if and only if it is (*efficiently*) computable by a Turing machine.

Quantum computing is a candidate that disproves the *extended* Church-Turing thesis

5. Exercise 1

Let n be a composite number. It is then easy to see that n has a factor at most $\sqrt{n}$. Does this fact give us an efficient algorithm for factoring?

6. Exercise 2

The decision version of the factoring problem asks if n has a factor less than k when given (n, k) as input. Show that the factoring problem is efficiently reducible to this decision version. That is, if there is an efficient algorithm for the decision version, there is also an efficient algorithm for the standard version.