Average Case Complexity Theory

Sriram Venkatesh
sriramvenkatesh739@gmail.com

July 17, 2023

What We Will Cover

1. Computational problems
2. Standard definitions and theorems of worst case complexity theory
3. Distributional problems
4. Reductions between distributional problems
5. distNP completeness
6. Bounded halting in distNP complete

What is Complexity Theory?

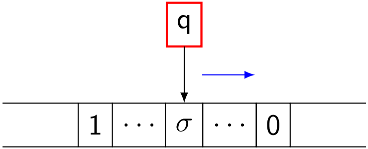
Classifying computational problems according to their resource usage into problem classes.

Examples:

1. Check if a list is sorted.
2. Add two numbers together.
3. Find the shortest path between two nodes in a graph.
4. Given a graph, check if it is two colorable.
5. Given a graph, check if it is three colorable.

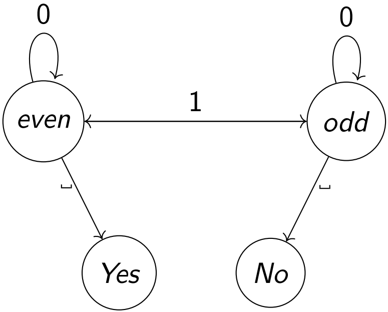
Turing Machines

- The fundamental model of computation we use to solve problems.
- A Turing machine consists of an infinite tape and a tape head, which can read, write, and move along the tape.



Turing Machines

Given an n bit string, check if the number of 1 bits are even.



1	1	0	1	0	␣	···

Types of Turing Machines

Definition

In a *deterministic Turing machine*, the set of rules prescribes at most one action to be performed for any given situation.

Types of Turing Machines

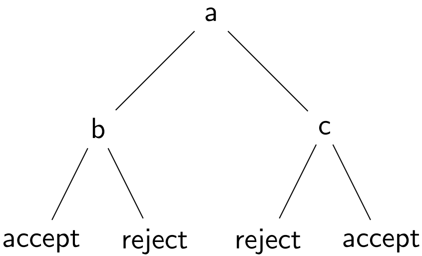
Definition

In a *deterministic Turing machine*, the set of rules prescribes at most one action to be performed for any given situation.

Definition

In a *nondeterministic Turing machine*, the set of rules may prescribe more than one action to be performed for any given situation.

Nondeterministic Turing Machines



Classes P vs NP

Definition

The class P consists of all computational problems that can be solved by a deterministic Turing machine in polynomial time.

Definition

The class NP consists of all computational problems that can be solved by a nondeterministic Turing machine in polynomial time.

- P vs NP problem.

Problems in class P

- Connected graph
- Three sum
- Two colorable

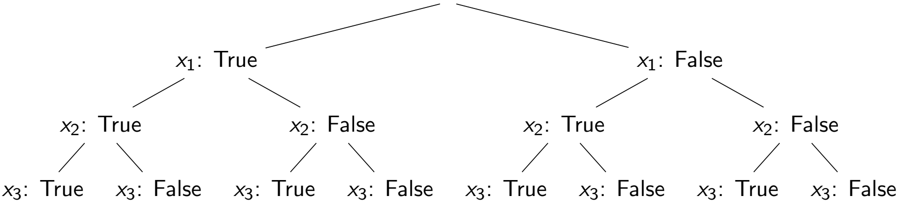
Boolean Satisfiability Problem

Definition

A *boolean formula* is built from boolean variables, operators AND ($\wedge$), OR ($\vee$), NOT (negation, $\neg$), and parentheses. A formula is said to be satisfiable if it can be made TRUE by assigning boolean values to its variables. The Boolean satisfiability problem (SAT) is, given a formula, to check whether it is satisfiable.

Example formula: $(x_1 \vee x_2) \wedge (\neg x_1 \wedge x_3)$
Assignment: x_1 : False; x_2 : True; x_3 : True

Boolean Satisfiability Problem



Polynomial Time Reductions

Definition

Problem A is reducible to problem B if an algorithm for solving problem B efficiently could also be used as a subroutine to solve problem A efficiently.

Polynomial Time Reductions

Definition

Problem A is reducible to problem B if an algorithm for solving problem B efficiently could also be used as a subroutine to solve problem A efficiently.

- Used to establish relationships between computational problems.
- We can now solve one problem by solving another problem.

NP complete problems

Definition

A problem is *NP* *complete* if it is in *NP* and every other *NP* problem is reducible to it.

NP complete problems

Definition

A problem is *NP complete* if it is in *NP* and every other *NP* problem is reducible to it.

Theorem

Cook Levin Theorem: The boolean satisfiability problem is NP complete.

Average Case Complexity Problems

Definition

An average case complexity problem consists of a problem D and a probability distribution μ, written as (D, μ).

Average Case Complexity Problems

Definition

An average case complexity problem consists of a problem D and a probability distribution μ, written as (D, μ).

- $\mu'(x) = \mu(x) - \mu(x-1)$
- Inputs to distributional problems are always binary numbers. Any efficient ordering of binary numbers is viable, although we will use standard lexicographic ordering of binary numbers.

A Problem with Traditional Definitions

Let us say an algorithm is efficient if it runs in expected polynomial time.

A Problem with Traditional Definitions

Let us say an algorithm is efficient if it runs in expected polynomial time.

- Let algorithm A run in time $O(2^n)$ on $\frac{1}{2^n}$ of the inputs, and run in $O(n^2)$ on $1 - \frac{1}{2^n}$ of the inputs.

A Problem with Traditional Definitions

Let us say an algorithm is efficient if it runs in expected polynomial time.

- Let algorithm A run in time $O(2^n)$ on $\frac{1}{2^n}$ of the inputs, and run in $O(n^2)$ on $1 - \frac{1}{2^n}$ of the inputs.
- Let problem B have running time $t_a(x)^2$. i.e. $O(2^{2n})$ on $\frac{1}{2^n}$ of the inputs and $O(n^4)$ on $1 - \frac{1}{2^n}$ of the inputs.

A Problem with Traditional Definitions

Let us say an algorithm is efficient if it runs in expected polynomial time.

- Let algorithm A run in time $O(2^n)$ on $\frac{1}{2^n}$ of the inputs, and run in $O(n^2)$ on $1 - \frac{1}{2^n}$ of the inputs.
- Let problem B have running time $t_a(x)^2$. i.e. $O(2^{2n})$ on $\frac{1}{2^n}$ of the inputs and $O(n^4)$ on $1 - \frac{1}{2^n}$ of the inputs.
- $\mathbb{E}(t_a(x)) = O(n^2)$.

A Problem with Traditional Definitions

Let us say an algorithm is efficient if it runs in expected polynomial time.

- Let algorithm A run in time $O(2^n)$ on $\frac{1}{2^n}$ of the inputs, and run in $O(n^2)$ on $1 - \frac{1}{2^n}$ of the inputs.
- Let problem B have running time $t_a(x)^2$. i.e. $O(2^{2n})$ on $\frac{1}{2^n}$ of the inputs and $O(n^4)$ on $1 - \frac{1}{2^n}$ of the inputs.
- $\mathbb{E}(t_a(x)) = O(n^2)$.
- $\mathbb{E}(t_b(x)) = O(2^n)$.

A Problem with Traditional Definitions

Let us say an algorithm is efficient if it runs in expected polynomial time.

- Let algorithm A run in time $O(2^n)$ on $\frac{1}{2^n}$ of the inputs, and run in $O(n^2)$ on $1 - \frac{1}{2^n}$ of the inputs.
- Let problem B have running time $t_a(x)^2$. i.e. $O(2^{2n})$ on $\frac{1}{2^n}$ of the inputs and $O(n^4)$ on $1 - \frac{1}{2^n}$ of the inputs.
- $\mathbb{E}(t_a(x)) = O(n^2)$.
- $\mathbb{E}(t_b(x)) = O(2^n)$.
- This is a problem!

Levin's Definition of Efficient on Average

Definition

An algorithm is said to have running time polynomial on the average if

$$\sum_{x \in \{0,1\}^*} \mu'(x) \frac{t(x)^\varepsilon}{|x|} = k,$$

where $t(x)$ represents the running time of the algorithm.

Levin's Definition of Efficient on Average

Definition

An algorithm is said to have running time polynomial on the average if

$$\sum_{x \in \{0,1\}^*} \mu'(x) \frac{t(x)^{\varepsilon}}{|x|} = k,$$

where $t(x)$ represents the running time of the algorithm.

- $\mathbb{E}(\frac{t_a(x)^{\varepsilon}}{|x|}) = k$

Levin's Definition of Efficient on Average

Definition

An algorithm is said to have running time polynomial on the average if

$$\sum_{x \in \{0,1\}^*} \mu'(x) \frac{t(x)^{\varepsilon}}{|x|} = k,$$

where $t(x)$ represents the running time of the algorithm.

- $\mathbb{E}(\frac{t_a(x)^{\varepsilon}}{|x|}) = k$
- $\mathbb{E}(\frac{t_b(x)^{\frac{\varepsilon}{2}}}{|x|}) = \mathbb{E}(\frac{(t_a(x)^2)^{\frac{\varepsilon}{2}}}{|x|}) = k$

Reductions Between Distributional Problems

- We reduce problem (D_1, μ_1) to (D_2, μ_2).
- Input x for D_1 is mapped to input $M(x)$ for D_2.
- What happens if $\mu'_1(x) >> \mu'_2(M(x))$?
- Solving (D_2, μ_2) does not mean we can solve (D_1, μ_1).

Reductions Between Distributional Problems

Efficiency, Validity, and Domination

Reductions Between Distributional Problems

Efficiency, Validity, and Domination

- Domination: High probability inputs map to high probability inputs, and low probability inputs map to low probability inputs.

$$\sum_{x \in \{0,1\}^*} \mathbb{P}[M(x) = y] \cdot \mu_1'(x) \leq \mu_2'(y) \cdot |y|^c,$$

Usefulness of Reductions

Theorem

If (D_1, μ_1) is reducible to (D_2, μ_2) through a deterministic polynomial time oracle Turing machine and (D_2, μ_2) is solvable by a deterministic Turing machine with a polynomial on average running time, then so is (D_1, μ_1).

Bounded Halting Problem

Halting Problem

Given a program and an input, will the program terminate?

Bounded Halting Problem

Halting Problem

Given a program and an input, will the program terminate?

Bounded Halting Problem

Given a program, an input, and k, will the program halt within k steps?

Bounded Halting Problem

Theorem

Bounded halting problem(BH) is NP complete.

Bounded Halting Problem

Theorem

Bounded halting problem(BH) is NP complete.

Proof.

A generic NP problem asks whether a nondeterministic Turing machine M accepts an input x in polynomial time. Pass in $(M, x, |x|^k)$ for some arbitrarily large constant k as input into bounded halting problem to solve the generic NP problem. ■

Bounded Halting Problem

Theorem

Bounded halting problem is distributional NP complete.

Bounded Halting Problem

Theorem

Bounded halting problem is distributional NP complete.

- $\mu'_{BH}(M, x, 1^k) = \frac{1}{|M|^2 \cdot 2^{|M|}} \cdot \frac{1}{|x|^2 \cdot 2^{|x|}} \cdot \frac{1}{k^2}$

Bounded Halting Problem

Theorem

Bounded halting problem is distributional NP complete.

- $\mu'_{BH}(M, x, 1^k) = \frac{1}{|M|^2 \cdot 2^{|M|}} \cdot \frac{1}{|x|^2 \cdot 2^{|x|}} \cdot \frac{1}{k^2}$
- Given an input x to a generic NP problem, we need to compress x into $c(x)$ such that the domination condition is satisfied

Bounded Halting Problem

Theorem

Bounded halting problem is distributional NP complete.

- $\mu'_{BH}(M, x, 1^k) = \frac{1}{|M|^2 \cdot 2^{|M|}} \cdot \frac{1}{|x|^2 \cdot 2^{|x|}} \cdot \frac{1}{k^2}$
- Given an input x to a generic NP problem, we need to compress x into $c(x)$ such that the domination condition is satisfied
- Compression $c(x)$ is the prefix of $\mu(x)$ which differentiates $\mu(x)$ from $\mu(x-1)$.

Bounded Halting Problem

Theorem

Bounded halting problem is distributional NP complete.

- $\mu'_{BH}(M, x, 1^k) = \frac{1}{|M|^2 \cdot 2^{|M|}} \cdot \frac{1}{|x|^2 \cdot 2^{|x|}} \cdot \frac{1}{k^2}$
- Given an input x to a generic NP problem, we need to compress x into $c(x)$ such that the domination condition is satisfied
- Compression $c(x)$ is the prefix of $\mu(x)$ which differentiates $\mu(x)$ from $\mu(x-1)$.

$$\begin{aligned} \mu(x-1) &= 0.101101010010 \\ \mu(x) &= 0.101101011110 \\ \mu(x+1) &= 0.101101101010 \end{aligned}$$

- Pass in $(M', c(x), |x|^k)$ as input to BH.

Bounded Halting Problem

Proof.

The size of $c(x)$ is at most $\log_2\left(\frac{1}{\mu'(x)}\right)$. The reduction from x to $c(x)$ is efficient due to the distributions being polynomially computable. The reduction is valid as well. For the domination condition, we have

$$\begin{aligned}
\mu'_{BH}(M', c(x), 1^{|x|^k}) &= \frac{1}{|M'|^2 \cdot 2^{|M'|}} \cdot \frac{1}{|c(x)|^2 \cdot 2^{|c(x)|}} \cdot \frac{1}{|x|^{k^2}} \\
&\leq k \cdot \frac{1}{c(x)^2} \cdot \frac{1}{|x|^{k^2}} \cdot \mu'(x) \\
&\leq \frac{1}{\text{poly}(|(M, c(x), 1^{|x|^k})} \cdot \mu'(x).
\end{aligned}$$

■

Things We Don't Know

- Completeness of classical *NP* complete problems i.e. interesting problems under interesting distributions
- Reductions between classical problems extended to their distributional analogues.