Intractable Problems

Notes on Automata and Theory of Computation

Chia-Ping Chen

Department of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan ROC

Introduction

- Whether there exists a TM for a language draws a line between RE and non-RE languages.
- Whether there exists an always-halting TM for a language draws a line between recursive and non-recursive languages.
- Being recursive may not be good enough as the time to finish computation may be unbearably long.
- The time complexity of an algorithm M is the maximum number of moves $T(n)$ (worst-case) needed for M to halt on an input of size n.
- $T(n)$ draws a line between tractable and intractable problems.

Polynomial-Time Algorithm

- An algorithm is said to be polynomial-time if $T(n)$ is a polynomial function of the input size n.
- If an algorithm is not polynomial-time, then it is sometimes referred to as being exponential.
- Note that the term *non-polynomial* is more exact than the term *exponential*, as there exist functions that are between polynomial and exponential.

$$p(n) = o(f(n)); f(n) = o(e^n).$$

$\mathcal{P}$ and Intractable Problems

- A problem Q is said to be in class $\mathcal{P}$ if there exists a *polynomial-time deterministic* algorithm that solves Q.
- Note that to solve a problem an algorithm must answer correctly for *any* instance of the problem.
- Since DTMs model real computers, any instance of a problem in $\mathcal{P}$ can be answered by a computer in polynomial time.
- A problem not in $\mathcal{P}$ is **intractable**, as its time complexity is more time than any polynomial function.

$$T(n) > n^k, \quad \forall k \in Z^+$$

Kruskal's Algorithm

- Consider the problem of finding a minimum-weight spanning tree (MWST) for a weighted graph.
- Kruskal's algorithm:
  - Initialization: each node is a *component* by itself. All edges are in set U. C is empty.
  - Iteration: retrieve the lowest-weight edge l from U. If it connects two components, merge them and put l in C.
  - Termination: when only a single component remains. Use C to construct the MWST.

Time-Complexity Analysis

- Suppose there are m nodes and e edges.
- In each epoch of iteration,
  - $O(e)$ to find the minimum-weight edge l in U
  - $O(m)$ to find the components for l
  - $O(m)$ to merge the components
- The algorithm finishes in e epochs, since the total number of components reduce by 1 for each epoch.
- The time complexity is $O(e(e+2m))$, polynomial in m, e.

Encoding for MWST Problem

- We can turn the MWST problem to an equivalent yes-no question: Is there a spanning tree with weight less than W?
- An encoding scheme for a weighted graph and W.

  $$100, 101000(1, 10, 1111)(1, 11, 1010)(10, 11, 1100)(10, 100, 10100)$$

  where $m = 4$, $W = 40$, and $(1, 10, 1111)$ represents an edge of weight 15 from node 1 to node 2. Most bits are used in the representation of edges.
- For an input string of length n, $e = O(n/\log m)$. In addition $m = O(e)$ for a connected graph. So

  $$T(n) = O(e(e + 2m)) = O(e^2) = O(n^2).$$

Non-deterministic TM and $\mathcal{NP}$

- The exact functional form of $T(n)$ depends on the TM used in the computation. However, as far as being polynomial or not is concerned, the various deterministic models are the same.
- The real distinction, for the matter of intractability, is between deterministic and non-deterministic TMs.
- A problem R is in the class $\mathcal{NP}$, if R can be solved (any instance of R can be answered correctly) by an NTM in some polynomial time.

Traveling Salesman Problem

- A salesman of city C is planning a tour to visit every city in a list with minimum cost, without visiting any city twice except for C.
- A set of edges that connects all nodes in a graph into a simple cycle is called a Hamilton circuit. TSP is related to the problem of Hamilton circuit.
- We are given a weighted graph and we ask if there exists a Hamilton circuit with weight less than W.
- TSP is $\mathcal{NP}$. With a non-deterministic TM, we can guessed an order (permutation) of the nodes and check if the cost is below W.

$\mathcal{P}$ and $\mathcal{NP}$

- By definition,

$$\mathcal{P} \subseteq \mathcal{NP}.$$

- An open question in computation theory is

$$\mathcal{P} \stackrel{?}{=} \mathcal{NP}.$$

- It is strongly believed that

$$\mathcal{P} \neq \mathcal{NP}.$$

  In other words, there are problems in $\mathcal{NP}$ that cannot be solved in polynomial time by a deterministic TM.

Polynomial-Time Reduction

- The method of reduction can be used in the development of theory of intractability.
- Here we reduce a problem Q_1 to Q_2 to show that Q_2 is at least as intractable as Q_1.
- Specifically, if Q_1 is not in $\mathcal{P}$, and every instance of Q_1 reduces in polynomial time to an instance of Q_2 with the same answer, then Q_2 cannot be in $\mathcal{P}$.
- Note that the reduction algorithm must be deterministic and polynomial-time.

$\mathcal{NP}$-Complete

- A problem Q is said to be $\mathcal{NP}$**-complete** if
  - $Q \in \mathcal{NP}$.
  - Every $Q' \in \mathcal{NP}$ is polynomial-time reducible to Q.
- If one solves an $\mathcal{NP}$-complete problem, say Q, then every problem in $\mathcal{NP}$ can be solved within a polynomial time of solving Q.
- Put in another way, an $\mathcal{NP}$-complete problem has the highest time complexity in $\mathcal{NP}$.
- If (as we believe) $\mathcal{P} \neq \mathcal{NP}$, all $\mathcal{NP}$-complete problems are in $\mathcal{NP} - \mathcal{P}$. Showing a problem to be $\mathcal{NP}$-complete is showing it to be intractable.

$\mathcal{NP}$-Complete and $\mathcal{P}$

- If some $\mathcal{NP}$-complete problem is in $\mathcal{P}$, then

$$\mathcal{P} = \mathcal{NP}.$$

  This follows since all problems in $\mathcal{NP}$ can be solved within a polynomial time.

- If some $\mathcal{NP}$-complete problem is not in $\mathcal{P}$, then

$$\mathcal{P} \neq \mathcal{NP}.$$

  This follows from the definition of $\mathcal{NP}$-completeness.

$\mathcal{NP}$-Hard

- A problem Q may appear to be so hard that we are not sure whether Q is in $\mathcal{NP}$.
- We may be able to find an $\mathcal{NP}$-complete problem Q_1 that reduces to Q. Then Q is not simpler than any $\mathcal{NP}$-complete problems.
- A problem like Q is said to be $\mathcal{NP}$-hard.
- Formally, if every $Q_1 \in \mathcal{NP}$ is polynomial-time reducible to Q, then Q is $\mathcal{NP}$-hard.

Co-$\mathcal{NP}$

- A language L is in Co-$\mathcal{NP}$ if its complement $\overline{L}$ is in $\mathcal{NP}$.
- If L is in $\mathcal{NP}$, then by definition $\overline{L}$ is in Co-$\mathcal{NP}$.
- If $\mathcal{P} = \mathcal{NP}$, then

$$\mathcal{P} = \mathcal{NP} = \text{Co-}\mathcal{NP}.$$

  - If L is in $\mathcal{P}$, then $\overline{L}$ is also in $\mathcal{P}$, and therefore in $\mathcal{NP}$. So L is in Co-$\mathcal{NP}$.
  - If L is in Co-$\mathcal{NP}$, then $\overline{L}$ is in $\mathcal{NP} = \mathcal{P}$. So L is in $\mathcal{P}$.

An $\mathcal{NP}$-Complete Problem

- In the theory of undecidability, the universal language L_u plays a fundamental role. To show a problem Q to be undecidable, we reduce L_u to Q. Indeed, we reduce L_u to MPCP to PCP to show PCP is undecidable.
- A fundamental problem that is $\mathcal{NP}$-complete is the SAT problem: Can a given Boolean expression be true for some assignment of variable values?
- Once we establish SAT to be $\mathcal{NP}$-complete, we can reduce SAT to a problem Q to show Q is $\mathcal{NP}$-complete.

SAT Problem

- A **Boolean expression** is built from
  1. Boolean variables
  2. binary operators $\wedge$ and $\vee$ for AND and OR
  3. unary operator $\neg$ for NOT
  4. parentheses
- A **truth assignment** for E assigns either true (1) or false (0) to each variable in E.
- The **value** of a Boolean expression is either 0 or 1.
- A Boolean expression E is satisfiable if some truth assignment makes E true.

Representing SAT Instances

- We use the following code
  - The operators and parentheses are represented by themselves.
  - Rename the variables $x_1, x_2, \ldots$. A variable x_i is represented by x followed by a binary string for i.
- Note that the length of code is approximately the same as the number of positions in the expression, counting each occurrence of variable as one position. If the number of positions is m, then the length of code is approximately $m \log m$.

Cook's Theorem

- SAT is $\mathcal{NP}$-complete.
- Two things need to be proved. First, we need to show that SAT is in $\mathcal{NP}$.
- We construct an NTM N for SAT. N guesses the truth assignment T. If $E(T) = 1$, then accept.
- The second requirement is to show every problem in $\mathcal{NP}$ reduces to SAT with a polynomial time. We show the reduction explicitly.

Notation of Proof

- Suppose L is in $\mathcal{NP}$ and an NTM M accepts L in polynomial time $p(n)$.
- Without loss of generality, we can assume that M never writes a blank or moves left of its initial position.
- If M accepts w with $|w| = n$, then there exists a computation

$$\alpha_0 \vdash \alpha_1 \vdash \cdots \vdash \alpha_k, \; k \leq p(n).$$

  - α_0 is initial ID.
  - α_k contains a final state.
  - Each α_i consists of non-blanks only. It extends from the initial head position to the right.

Idea of Proof

- Each α_i can be written as a sequence of symbols $X_{i0} \dots X_{ip(n)+1}$. There is no need to consider $X_{ip(n)+2}$.
- To describe ID's in terms of Boolean variables, we use indicator variable y_{ijA} for $X_{ij} = A$.
- We are going to construct a Boolean expression that is satisfiable iff M accepts w in $p(n)$ moves.
- In addition, the satisfying truth assignment will be the one that tells the truth about the ID's. y_{ijA} is true in the satisfying truth assignment iff $X_{ij} = A$.

Representing ID's

- We represent an ID to position $p(n)$ even that may include a tail of blanks.
- Assume all computations continue for exactly $p(n)$ moves. We allow $\alpha \vdash \alpha$ if a computation finishes early.
- A polynomial-time computation is then represented by a matrix.
  - The number of cells is a polynomial.
  - The number of variables that represent each cell is bounded.

Construct Boolean Expression

- Denote E_{Mw} as the target expression we want to construct based on the computation of an NTM M on w.
- Overall

$$E_{Mw} = S \wedge N \wedge F,$$

where S, N, F are expressions that ensures that M starts, moves and finishes right.

Starts Right

- The initial ID is $q_0 w$ followed by blanks.
- Let $w = a_1 \ldots a_n$. Let

$$S = y_{00q_0} \wedge y_{01a_1} \wedge y_{02a_2} \cdots \wedge y_{0na_n} \wedge y_{0n+1B} \cdots \wedge y_{0p(n)B}$$

- S would be true only for the intended initial ID.

Finishes Right

- There is an accepting state in the final ID.
- Let $f_1 \ldots f_k$ be the final states. Define

$$F_j = y_{p(n)jf_1} \vee y_{p(n)jf_2} \vee \cdots \vee y_{p(n)jf_k},$$

which indicates whether the symbol in position j of ID $p(n)$, $X_{p(n)j}$, is an accepting state.

- Let

$$F = F_0 \vee F_1 \vee \cdots \vee F_{p(n)}.$$

F would be true as long as there is an accepting state in the last row.

Next Move Is Right

- This is the most complicated part.
- We construct N such that

$$N = N_0 \wedge N_1 \wedge \cdots \wedge N_{p(n)}$$

where

$$N_i = (A_{i0} \vee B_{i0}) \wedge (A_{i1} \vee B_{i1}) \wedge \cdots \wedge (A_{ip(n)} \vee B_{ip(n)})$$

- It will be shown that A_{ij} and B_{ij} together take care of correctness of position j in going from α_i to α_{i+1}.
- Observe that X_{i+1j} is determined by $X_{ij-1}, X_{ij}, X_{ij+1}$.

A_{ij}

- A_{ij} ensures that if X_{ij} is a state then the positions $j, j \pm 1$ are correct.
- Consider $X_{ij-1}X_{ij}X_{ij+1}$ and $X_{i+1j-1}X_{i+1j}X_{i+1j+1}$.
- If X_{ij} is a state, then $X_{i+1j-1}X_{i+1j}X_{i+1j+1}$ and $X_{ij}X_{ij+1}$ are related by the transition function: If $(p, C, L) \in \delta(q, A)$, then $\alpha\ DqA\ \beta \vdash \alpha\ pDC\ \beta$, so we want a clause

$$y_{ij-1D} \wedge y_{ijq} \wedge y_{ij+1A} \wedge y_{i+1j-1p} \wedge y_{i+1jD} \wedge y_{i+1j+1C}$$

  Similarly for $(p, C, R) \in \delta(q, A)$.
- A_{ij} is the OR of all valid terms.

B_{ij}

- Let $q_1 \dots q_m$ be the states of M, and $Z_1 \dots Z_r$ be the tape symbols.

$$
\begin{aligned}
B_{ij} = {} & (y_{ij-1q_1} \vee y_{ij-1q_2} \cdots \vee y_{ij-1q_m}) \vee \\
& (y_{ij+1q_1} \vee y_{ij+1q_2} \cdots \vee y_{ij+1q_m}) \vee \\
& ((y_{ijZ_1} \vee y_{ijZ_2} \vee \cdots \vee y_{ijZ_r}) \wedge \\
& ((y_{ijZ_1} \wedge y_{i+1jZ_1}) \vee (y_{ijZ_2} \wedge y_{i+1jZ_2}) \cdots \vee (y_{ijZ_r} \wedge y_{i+1jZ_r})))
\end{aligned}
$$

- B_{ij} is true if one of the positions $j \pm 1$ is a state. The correctness of X_{i+1j} will be taken care of by $A_{ij\pm1}$.
- B_{ij} is also true if none of the positions $j, j \pm 1$ is a state and $X_{i+1j} = X_{ij}$.

Conclusion of Cook's

- Note that the size of E_{Mw} is polynomial in $|w|$.
  - S has $p(n)$ variables.
  - F has $p(n)+1$ F_j's, and each F_j has k variables.
  - N has $p(n)$ N_i's, and each N_i has $p(n)+1$ $(A_{ij} \vee B_{ij})$'s. Each B_{ij} has $m+m+r+2r$ variables and each A_{ij} has $mr * 6$ variables.
- Since k, m, r are constants for given M, the size of E_{Mw} is in the order of $(p(n))^2$. Writing E_{Mw} is thus polynomial time.

Restricted SAT Problems

- We are going to use SAT to show that some well-known problems, e.g. TSP, are $\mathcal{NP}$-complete.
- We first consider restricted versions of SAT, called CSAT, k-SAT and 3SAT. We show SAT problem reduces to these problems in polynomial time.
- These problems reduce to the well-known problems, such as IS (independent set), NC (node cover), DHC (directed Hamilton circuit), HC.
- The reduction involves constructing specific graphs for certain SAT problems.

Conjunctive Normal Forms

- A **literal** is either a variable or a negated variable.
- A **clause** is the OR of one or more literals.
- We often use $^{-}$ for negation ($\neg$), $+$ for OR ($\vee$) and product for AND ($\wedge$).

$$(x+\bar{y})(yz) \;\Rightarrow\; (x \vee \neg y) \wedge (y \wedge z)$$

- A Boolean expression is said to be in **conjunctive normal form**, or **CNF**, if it is the AND of clauses.
- Specifically, it is in k-CNF if it is the product of clauses, each of which is the sum of k distinct literals.

CSAT and 3SAT

- CSAT: Given a Boolean expression in CNF, is it satisfiable?
- 3SAT: Given a Boolean expression in 3-CNF, is it satisfiable?
- Both problems are $\mathcal{NP}$-complete.
  - We first reduce SAT to CSAT in polynomial time.
  - We then reduce CSAT to 3SAT.

Converting to CNF

- Two Boolean expressions are *equivalent* if they have the same value for any truth assignment to their variables.
- In reducing SAT to CSAT, given instance E of SAT, our goal is to construct an instance F in CSAT such that E is satisfiable iff F is. It is not necessary that E and F are equivalent.
- There are two steps in this conversion.
  1. Construct E' that is equivalent to E.
  2. Construct F that is satisfiable iff E' is.

Construction of E'

- For any E in SAT, we can construct an equivalent E' such that the negation is on literals only. Furthermore, the length of E' is linear in the number of symbols in E, and E' can be constructed in polynomial time.
- This can be proved by mathematical induction with the help of *DeMorgan's laws*.

$$\neg(E \wedge F) \;\Rightarrow\; \neg(E) \vee \neg(F)$$
$$\neg(E \vee F) \;\Rightarrow\; \neg(E) \wedge \neg(F)$$
$$\neg(\neg(E)) \;\Rightarrow\; E$$

- Try an example would be convincing.

Construction of F

- If E' as previously defined is of length n (the number of positions), then there is an F such that
  - F is in CNF, with at most n clauses.
  - F can be constructed from E' in time cn^2.
  - A truth assignment T' for E' makes E' true iff an extension S of T' makes F true.
- The tricky part of proof is the inductive case where $E' = E'_1 \vee E'_2$. Note $F_1 \vee F_2$ is not in CNF. Let $F_1 = g_1 \wedge \cdots \wedge g_p$ and $F_2 = h_1 \wedge \cdots \wedge h_q$. Introduce a variable y and define F in CNF

$$F = (y + g_1) \wedge \cdots \wedge (y + g_p) \bigwedge (\bar{y} + h_1) \wedge \cdots \wedge (\bar{y} + h_q)$$

Proof

- Suppose T' makes E' true. Then an extension S of T' including y makes F true:
  - Either E'_1 or E'_2 is true.
  - If E'_1 is true, then an extension S_1 of T'_1 makes F_1 true (by inductive assumption). Then assigning $y = 0$ makes F true.
  - Similarly for the case E'_2 is true.
- Suppose S satisfies F. Then there exists a T' where S is an extension of T' satisfies E' .
  - If $y = 0$, F_1 must be true. S_1 exists and T'_1 for E'_1 exists by inductive assumption.
  - Similarly for the case $y = 1$.

CSAT to 3SAT

- We can convert an expression $E = e_1 \wedge \cdots \wedge e_m$ in CNF to one in 3-CNF as follows.
  - If $e_i = (x)$ is single, replace e_i by
  $$x \longrightarrow (x+u+v)(x+u+\bar{v})(x+\bar{u}+v)(x+\bar{u}+\bar{v})$$
  - If $e_i = (x+y)$ contains two literals, replace e_i by
  $$x+y \longrightarrow (x+y+\bar{z})(x+y+z)$$
  - If $e_i = (x_1 + \cdots + x_m)$, replace e_i by
  $$(x_1+x_2+y_1)(x_3+\overline{y_1}+y_2)(x_4+\overline{y_2}+y_3)\ldots(x_{m-1}+x_m+\overline{y_{m-3}})$$

More $\mathcal{NP}$-Complete Problems

- With 3SAT, we now show that some problems in graphs are $\mathcal{NP}$-complete.
- Description for $\mathcal{NP}$-complete problems.
  - *name*
  - *input*
  - *output*
  - *reduce from*
- Example
  - *CSAT*
  - *a Boolean expression in CNF*
  - *Yes, if satisfiable*
  - *SAT*

Independent Set Problem

- An independent set of a graph $G = (V, E)$ is a subset I of V such that no nodes in I is connected by an edge.
- The independent set problem is described by
  - *IS*
  - *a graph G and a lower bound k*
  - *Yes, if G has an independent set of k nodes*
  - 3*SAT*
- We need to construct an instance (G, k) of IS based on an instance E of 3SAT.

Reducing 3SAT to IS

- Given an E in 3-CNF with m clauses, we construct a graph G such that
  - For a clause in E we create a clique of three nodes, with each node representing a literal.
  - There is an edge between the node for a literal and the node for its complement.
- An independent set of size m in G indicates E is satisfiable by setting the literals of the nodes in the IS true.
  - One node in each clique is chosen.
  - A literal and its complement cannot be chosen simultaneously.

Node-Cover Problem

- A node cover of a graph $G = (V, E)$ is a subset C of V such that each edge in E has at least one of its end nodes in C.
- The node-cover problem is described by
  - *NC*
  - *a graph G and a lower bound k*
  - *Yes, if G has a node cover of k or fewer nodes*
  - *IS*
- Note that the complement of a node cover is an independent set. The reduction of an instance of IS to an instance in NC is merely the change k in IS to $n - k$ in NC. A graph has a node cover of $n - k$ nodes iff it has an independent set of k nodes.

Directed Hamilton Circuit

- A Hamilton circuit in a directed graph $G = (V, E)$ is a directed simple cycle that connects all nodes.
- The directed Hamilton circuit problem is described by
  - *DHC*
  - *a directed graph* G
  - *Yes, if* G *has a directed Hamilton circuit*
  - 3*SAT*

Reducing 3SAT to DHC

- Suppose E is a k-clause 3-CNF with n variables.
  - For each variable x_i we construct a subgraph H_i, which is shown in Figure 10.9(a). a_i, d_i are the entry/exit nodes. b_{ij}, c_{ij} are nodes designed to indicate whether $x_i = 1$ or 0.
  - For each clause e_j we will have a subgraph I_j, which is shown in Figure 10.9(c). Note a cycle must enter and leave in the same column.
  - H_i and I_j are connected as shown in Fig 10.10: For x_i in e_j we pick an unused c_{ip} for r_j and b_{ip+1} for u_j. Likewise, for $\overline{x_i}$ in e_j we pick an unused b_{ip} for r_j and c_{ip+1} for u_j.
- A DHC in G indicates E is satisfiable by setting the variables of the H_i's accordingly.

Hamilton Circuit Problem

- A Hamilton circuit in an undirected graph $G = (V, E)$ is a simple cycle that connects all nodes.
- The Hamilton circuit problem is described by
  - *HC*
  - *an undirected graph G*
  - *Yes, if G has a Hamilton circuit*
  - *DHC*
- The reduction goes as follows. Given an instance G for DHC, we construct an instance G' of HC. Each node v in G corresponds to 3 nodes v^0, v^1, v^2 in G', where v^0, v^1 and v^1, v^2 are connected. For a directed edge (u, w) in G, we have undirected edges (u^2, w^0) in G'.

Traveling Salesman Problem

- Description
  - *TSP*
  - *an undirected graph G with weights on the edges, and a limit k*
  - *Yes, if there is a Hamilton circuit such that the sum of edges is less or equal to k*
  - *HC*
- The reduction goes as follows. Given an instance G for HC we construct an instance G' of TSP that is exactly like G except that we assign the weight on every edge to be 1. Solving G' of TSP for $k = n$ solves G of HC.

Randomized TM

- Sometimes we need random numbers in an algorithm. For example, in *quick sort*, the pivot can be chosen randomly.
- How do we implement randomization with a TM?
- Answer: we can use an extra tape which stores random bits, with each bit being 1 with probability $1/2$. It is equivalent to flipping a fair coin at every move.
- A randomized TM is given in Figure 11.7. $qUVDE$ means the TM enters state q, writes symbols U, V, and moves in the directions D, E for the input tape and random tape.

Monte-Carlo TM

- For a randomized TM, acceptance and the run time becomes random.
- The language L of a Monte-Carlo TM M is defined by
  - Every w is in L is accepted by M with probability at least $1/2$.
  - If w is not in L, then M does not accept w with probability 1.
- In one simulation, if x is accepted by M, then $x \in L$. If x is not accepted by M, the x may or may not be in L.
- In other words, there is a chance for false rejection for $w \in L$ but not a chance for false acceptance for $w \notin L$.

Class $\mathcal{RP}$

- A language L is in class $\mathcal{RP}$ (*Random Polynomial*) if
  - L is accepted by a MC TM M.
  - There is a polynomial $T(n)$ such that for any input w of size n, M halts in no more than $T(n)$ steps for any simulation.
- M is also called a polynomial-time MC algorithm.

Las-Vegas TM

- A randomized TM that always halts and gives the correct answer is called a Las-Vegas TM.
- That is, for an LV TM M, the acceptance event does not depend on the random tape content. There is no chance of error.
  - $w \in L(M)$ is accepted by M with probability 1.
  - $w \notin L(M)$ is rejected by M with probability 1.

Class $\mathcal{ZPP}$

- The expected time of M to halt on input w depends on w.
- We define another class of languages by the expected time of computation.
- A language L is in class $\mathcal{ZPP}$ (*Zero-error, Probabilistic, Polynomial*) if
  - L is accepted by a LV TM M.
  - There is a polynomial $T(n)$ such that for any input w of size n, the expected time for M to halt on w is no more than $T(n)$.

$\mathcal{P} \subseteq \mathcal{ZPP}$

- It can be shown that

$$\mathcal{P} \subseteq \mathcal{ZPP} \subseteq \mathcal{RP} \subseteq \mathcal{NP}$$

- If L is in $\mathcal{P}$, then L is accepted by a deterministic TM M. M is a special case of randomized TM which treats random bits 0 and 1 equally. So L is in $\mathcal{ZPP}$.

$\mathcal{ZPP} \subseteq \mathcal{RP}$

- If L is in $\mathcal{ZPP}$, then L is accepted by a LV TM M with expected time $T(n)$.
- We can construct a TM N from M: N simulates the computation of M for $2T(n)$ steps. If M accepts, then N accepts.
  - If $w \in L$, then N accepts with probability at least $1/2$.
  - If $w \notin L$, then N accepts with probability 0.
- N is a MC TM for L that halts in $2T(n)$ steps. So L is in $\mathcal{RP}$.

$\mathcal{RP} \subseteq \mathcal{NP}$

- Suppose L in $\mathcal{RP}$, then there is a polynomial-time MC TM M for L.
- We can construct an NTM N from M. Whenever a random bit of M is scanned for the first time, N chooses one of two alternatives corresponding to bit 0 or 1 nondeterministically and writes the bit on its tape. Then N simulates the running of M.
- If $w \in L$, then w is accepted by M, and also by N. If $w \notin L$, then w is not accepted by M, and also not by N.