Randomised algorithms and Probabilistic complexity classes

1) Computational model

Probabilistic Turing machine: Deterministic Turing machine augmented with the possibility of sampling a random bit uniformly at random.

Alternatively, a deterministic Turing machine, taking in entry an instance and a random infinite sequence of bits.

2) Las Vegas algorithms

Las Vegas algorithms: the running time depends on the random bits, but the answer is sure:

We typically attempt to bound the expectation of the running time.

Example: to sort a list of elements, the quick sort algorithm has complexity n^2: if the chosen pivot is the smallest element, than it take $O(n^2)$ steps.

But if we choose it at random, then

$$\exists C,\ \mathbb{P}(\text{runningtime} \geq Cn\log_n) = 1 - o(1).$$

Lemma: $\forall C_1, \exists C_2$ such that:

Let T be a complete binary tree of depth n, for every node $u \in V(T)$, u is good with probability $\frac{1}{2}$.

$$\mathbb{P}(\forall v \text{ at depth} \geq C_2 \log n, v \text{ has at least } C_1 \log n \text{ good ancestors}) = 1 - o(1)$$

Proof: Let v be a node at depth $C_2 \log n$.

Let X_v count the number of good ancestors of v.

$$\mathbb{E}[X_v] = \frac{C_2}{2} \log n. \qquad \text{By Chernoff bound,}$$

$$\mathbb{P}(X_v \leq C_1 \log n) \leq \exp\left(-\frac{2\left(C_1 - \frac{C_2}{2}\right)^2 \log^2 n}{3\, C_2 \log n}\right)$$

$$\leq O\left(n^{-\frac{2C_1}{3C_2} + \frac{1}{3}}\right)$$

The number of nodes at depth $C_2 \log n$ is $O(n^{C_2})$

By union bound,

$$\mathbb{P}(\forall v \text{ at depth} \geq C_2 \log n,\ X_v \geq C_1 \log n)$$

$$\leq 1 - O\left(n^{C_2} \cdot n^{-\frac{2C_1}{3C_2} + \frac{1}{3}}\right)$$

$$\leq 1 - o(1) \qquad \text{taking } C_2 = \frac{\sqrt{C_1}}{10}.$$

∎

In the execution of quicksort with random pivot, a node in the decision tree is good if:

- the current list L to sort is of size > 4, if the pivot creates two sublists $L_{<x}$ and $L_{>x}$ of size $\geq \frac{|L|}{4}$.
- If the current list L to sort is of size < 4, with probability $\frac{1}{2}$.

$$P(u \text{ is good}) = p = \frac{2}{3}.$$

X = waiting time for $\ell = C_1 \log n$ good nodes.

$$P(X \geq C_2 \log n) = \sum_{k=0}^{\ell-1} \binom{C_2 \log n}{k} p^k (1-p)^{C_2 \log n - k}$$

$$= (1-p)^{C_2 \log n} \sum_{k=0}^{\ell-1} \binom{C_2 \log n}{k} \left(\frac{p}{1-p}\right)^k$$

$$\leq (1-p)^{C_2 \log n} \binom{C_2 \log n}{C_1 \log n} \sum_{k=0}^{\ell-1} \left(\frac{p}{1-p}\right)^k$$

if $\ell \leq \frac{C_2}{2} \log n$

$$\leq \frac{1}{3^{C_2 \log n}} \, 2 \binom{C_2 \log n}{C_1 \log n}$$

$\binom{n}{k} \leq \left(\frac{en}{k}\right)^k$

$$\leq 2 \cdot \frac{1}{3^{C_1 \log n}} \left(\frac{e C_2}{C_1}\right)^{C_1 \log n}$$

$$\leq 2 \cdot \left(\left(\frac{e C_2}{3^{\frac{C_2}{C_1}} C_1} \right)^{C_1} \right)^{\log n} = o\left(\frac{1}{n^2}\right)$$

for C_2 large enough

$$\leq 2 \, n^{\log(-)}$$

there are n leaves.

The algorithm terminates at the largest depth where there exist a node with less than $\log_4 n$ good ancestors.
So by lemma 1, the running time is $O(n \log n)$ with probability $1 - o(1)$.

Choosing a pivot at random is the same shuffling randomly the list before running a deterministic algorithm.

3) Monte Carlo algorithms:

A **Monte Carlo algorithm** is an algorithm that runs in deterministic time but whose answer is probably correct.

One-sided: $\mathbb{P}$(Algo answers correctly on negative instances) $= 1$
$\mathbb{P}$(Algo answers correctly on positive instances) $= 1 - \varepsilon$.

$\rightarrow$ No false negatives!

Two-sided: Both probabilities are $\geq 1 - \varepsilon_1$, $\geq 1 - \varepsilon_2$

3) Complexity classes

RP : Randomised polynomial time
→ One-sided Monte Carlo
→ runs in polynomial time.

ie. $\begin{cases} x \in L \Rightarrow P(A \text{ accepts } x) \geq \frac{1}{2} \\ x \notin L \Rightarrow P(A \text{ accepts } x) = 0 \end{cases}$

One can boost this algorithm and reduce the probability of error by repeating the algorithm, then $x \in L \Rightarrow P(A^{(t)} \text{ accepts } x) \geq 1 - \frac{1}{2^t}$.

co-RP : $\overline{L} \in RP$

ZPP: Zero-Error Probabilistic Polynomial time
→ Las Vegas algorithms running in polynomial expected time.

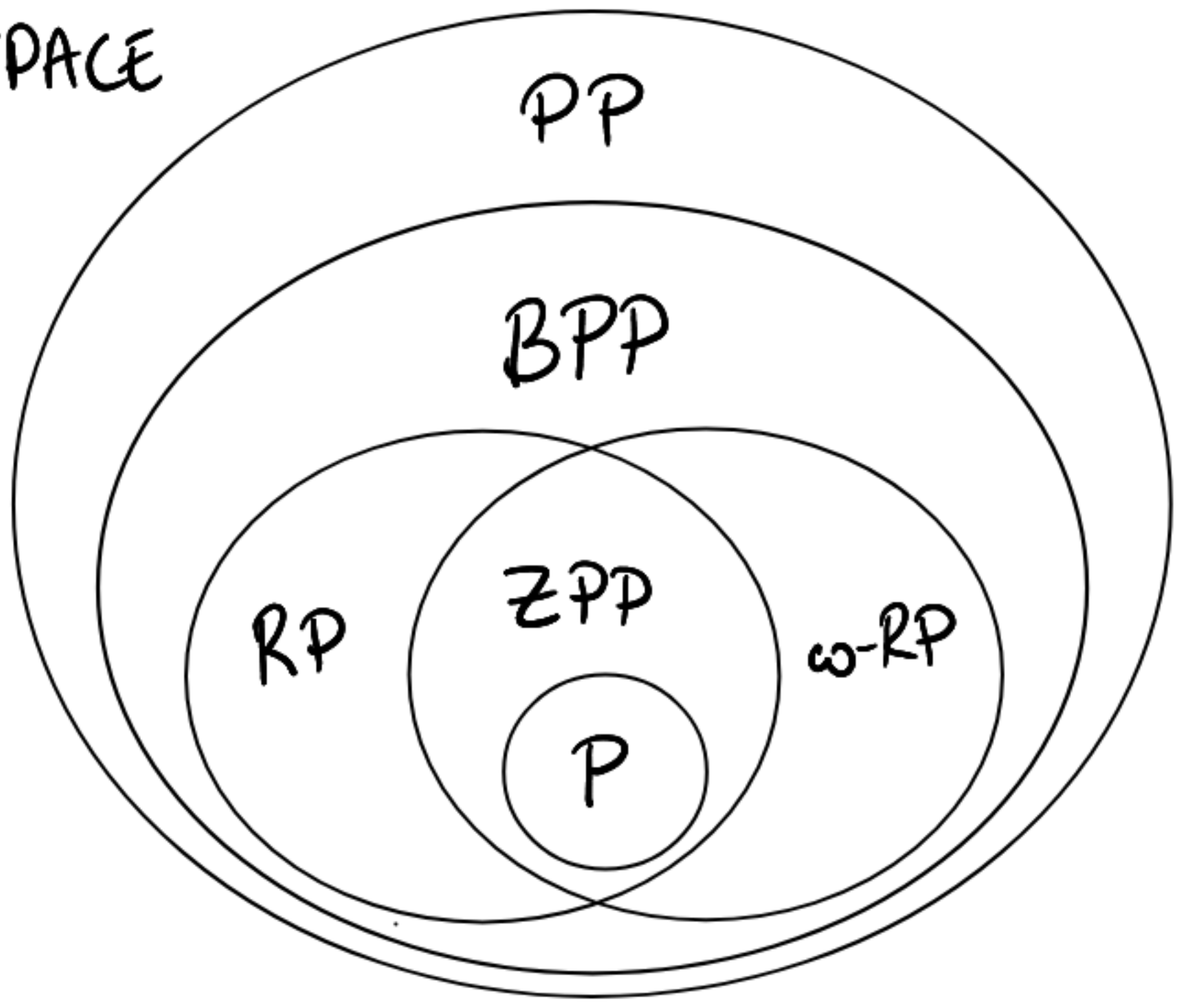
PP: Probabilistic polynomial time
→ two sided Monte Carlo
→ runs in polynomial time

s.t $\begin{cases} x \in L \Rightarrow P(A \text{ accepts } x) > \frac{1}{2} \\ x \notin L \Rightarrow P(A \text{ accept } x) < \frac{1}{2} \end{cases}$

Note: it's a very weak assumption we do not know how well separated these probabilities are from $\frac{1}{2}$, so we do not have a bound on the number of repetition needed to boost the success rate (e.g. it might be such polynomial).

BPP : Bounded probabilistic polynomial time.
→ two sided Monte Carlo
→ runs in polynomial time

s.t $\begin{cases} x \in L \Rightarrow P(A \text{ accepts } x) \geq \frac{3}{4} \\ x \notin L \Rightarrow P(A \text{ accepts } x) \leq \frac{1}{4}. \end{cases}$



- $P \subseteq RP \subseteq NP$:
Probabilistic execution can be simulated by non-determinism

- We don't know how BPP and NP compare.

- $NP \subseteq BPP$ is considered unlikely, because it would give practical solutions for NP-complete problems.

Sipser - Gács - Lautemann 1983 $BPP \subseteq \Sigma_2 \cap \Pi_2$.

so if $P = NP$, the PH collapses and $BPP = P$.

Thus $P \neq NP$ or $P = BPP$ (or both).

4) Approximation algorithms:

MAX-SAT: (NP-hard).

Entry: a CNF-formula on n variables on m clauses.
Outputs: maximum number of satisfied clauses in a truth assignment.

Theorem: For every k-CNF formula with m clauses, there is truth assignment satisfying at least $\left(1-\frac{1}{2^k}\right)m$ of them.

Proof: Set each variable to true or false independently at random.

$$\mathbb{P}(C_i \text{ is not satisfied}) = \frac{1}{2^k}.$$

$$\mathbb{E}[\text{satisfied clauses}] \underset{L.E}{=} \frac{m}{2^k}$$

By F.M., $\mathbb{P}\left(\text{at least } \left(1-\frac{1}{2^k}\right)m \text{ clauses are satisfied}\right) > 0.$ ▨

A deterministic / randomised algorithm $\mathcal{A}$ is a (probabilistic) α-approx for a maximisation problem if for every instance I,

$$\mathbb{E}[\mathcal{A}(I)] \geq \alpha \cdot \text{opt}(I)$$

Observation: we just gave a probabilistic $(1-2^{k-1})$-approx for MAXSAT on k-CNF formulas.

How to get rid of the k-dependency?

Small clauses are harder to satisfy.

Theorem: There exist a polynomial time probabilistic $(1-1/e)$-approx for MAXSAT:

Proof: We use a linear program. Let φ be a CNF formula on n variables and m clauses (without restrictions on the number of literals per clause).

Let $x_1, \dots x_n$ be the variables of φ.

For every $i \in [n]$, let y_i be a $\{0,1\}$ variable

For every $j \in [m]$, let z_j be a $\{0,1\}$ variable.

$$\max \sum_{j=1}^{m} z_j \quad \text{under} \quad \forall j, \sum_{x_i \in C_j} y_i + \sum_{\overline{x_i} \in C_j} (1 - y_i) \geq z_j$$

The solution of this ILP is $opt_{MAXSAT}(\varphi)$.

We solve its relaxation an obtain a solution
$(\hat{y}_1, \dots \hat{y}_n, \hat{z}_1, \dots \hat{z}_m) \in [0,1]^{n+m}$.

Clearly $\sum \hat{z}_j \geq opt_{MAXSAT}(I)$.

For every i, assign x_i to $\begin{cases} \text{True with probability } \hat{y}_i \\ \text{False} \quad \text{---------} \quad (1-\hat{y}_i \end{cases}$

Consider a clause C_j. W.l.o.g, write $C_j = (x_1 \vee x_2 \dots \vee x_k)$ for some k.

$$P(C_j \text{ is True}) = 1 - \prod_{i=1}^{k} (1 - \hat{y}_i) \qquad (1)$$

We have $\hat{y}_1 + \dots \hat{y}_k \geq \hat{z}_k$ so

$$\mathbb{P}(C_j \text{ is true}) = 1 - \prod_{i=1}^{k} (1 - \hat{y}_i)$$

$$\geq 1 - \left(1 - \frac{\hat{z}_j}{k}\right)^k \qquad \text{because } \sum_{i=1}^{k} \hat{y}_i \geq \hat{z}_j$$

$f(z) = 1 - \left(1 - \frac{z}{k}\right)^k$ is concave and $f(0) = 0$, $f(1) = 1 - \left(1 - \frac{1}{k}\right)^k$

so $f(z) \geq f(1)\, z$ for $z \in [0,1]$.

Hence, for every clause C_j on k_j variable,

$$\mathbb{P}(C_j \text{ is true}) \geq \left[1 - \left(1 - \frac{1}{k}\right)^k\right] \hat{z}_j$$

So $\mathbb{E}[\text{number of satisfied clauses}] \geq_{\text{L.E.}} \min_{k_j} \underbrace{\left[1 - \left(1 - \frac{1}{k}\right)^k\right]}_{:= \beta_k} \cdot \text{opt}_{\text{MAXSAT}}(\varphi)$

As $\beta_k \geq 1 - 1/e \ \forall k$, we are done ∎

We can do even better:

k	$1 - 2^{-k}$	β_k
1	0.5	1.0
2	0.75	0.75
3	0.875	0.704
4	0.937	0.684

Theorem: there is a polynomial time $\frac{3}{4}$-approx.

We run both algorithms and take the best of the two.

Let m_1 be the solution returned by the first algorithm
m_2 ——————— second ——— .

$$\mathbb{E}[\max(m_1, m_2)] \geq \max(\mathbb{E}[m_1], \mathbb{E}[m_2]) \geq \frac{\mathbb{E}[m_1] + \mathbb{E}[m_2]}{2}$$

because m_1 and m_2 are independent

we want to show $\max[\mathbb{E}[m_1], \mathbb{E}[m_2]] \geq \frac{3}{4} \sum_j \hat{z}_j$.

Let S^k denote the set of clauses with k literals.

$$\mathbb{E}[m_1] = \sum_k \sum_{c_j \in S^k} (1 - 2^{-k})$$

$$\geq \sum_k \sum_{c_j \in S^k} (1 - 2^{-k}) \hat{z}_j$$

$$\text{So } \frac{\mathbb{E}[m_1] + \mathbb{E}[m_2]}{2} \geq \sum_k \sum_{c_j \in S^k} \underbrace{\frac{(1 - 2^{-k}) + \beta^2}{2}}_{\geq \frac{3}{4} \ \forall k} \hat{z}_j .$$

$\geq \frac{3}{2}$ $\forall k$.

∎