Advanced Algorithms

Spring 2021

Lecture 1

Agenda for Today

- Welcome back to a new semester !!
  - We are hopefully seeing the last embers of the pandemic.
  - Nevertheless, we have to prepare for a possibly new normal in many ways.
  - We will continue to teach and learn in the online mode for Spring 2021 too.
  - Stay safe, but curious!

Agenda for Today

- Welcome back to a new semester !!
  - We are hopefully seeing the last embers of the pandemic.
  - Nevertheless, we have to prepare for a possibly new normal in many ways.
  - We will continue to teach and learn in the online mode for Spring 2021 too.
  - Stay safe, but curious!

- My details
  - Kishore Kothapalli, Professor, IIIT Hyderabad
  - Email: kkishore@iiit.ac.in (the best way to reach me)
  - Research interests span parallel computing and distributed algorithms.

Agenda for Today

- Rest of Today's Class
  - Syllabus
  - Policies
  - Expectations
  - Actual lecture

Syllabus

- Roughly, a three module course
  - Module 1: Randomized Algorithms
  - Module 2: Parallel and Distributed Algorithms
  - Module 3: Advanced Topics
    - Big data and Sampling
    - Algorithm engineering
    - Any other

Syllabus

- Roughly, a three part course
  - Randomized Algorithms
    - Chernoff bounds and Randomized Routing
    - Perfect Hashing
    - Graph algorithms: MIS, Spanners
    - Randomized Rounding
    - Approximate counting
  - Parallel and Distributed Algorithms

Syllabus

- Roughly, a three part course
  - Randomized Algorithms
  - Parallel and Distributed Algorithms
    - Flynn's Taxonomy and Models of Computations
    - Basic parallel algorithms: Search/Sort/Scan/Merge
    - Tree and Graph Algorithms
    - Lower Bounds

Policies

- Grading (Tentative)
  - Homeworks: 30% (We will have some lateness policy here)
  - In-class Quizzes : 25% (We will have some redundancy here)
  - End Exam: 15%
  - Quizzes 1 and 2 : 30%
  - Exceptional Performance: 5% extra

- Any submission that is graded and evaluated should not be copied from any source.
- Copied submissions will get zero for the first instance and negative for repeat offences.

We have two Teaching Assistants named so far:

Support Staff

Sayantan Jana, sayantan.jana@research.iiit.ac.in
Athreya Chandramouli, athreya.chandramouli@research.iiit.ac.in

Policies

- Textbooks: Do not own these books just for the class!
  - Randomized Algorithms, Motwani and Raghavan
  - Introduction to Parallel Algorithms, J. JaJa
  - Other material to be posted on the course website

- Most welcome to write to me if you have any questions.

Expectations

- Utilize class time effectively.
  - Starts with all of you settling by the class time.
  - Ask any question you may have. No question is small to ask.
  - Do not show up late.

Agenda for Today

- On to the actual lecture….randomization in computing
- We will see how randomization can help in designing and analyzing algorithms.
- Starting with a very simple example...

A Simple Example

- Let us recall the partition procedure and quick sort.
- We assume that the elements of the set are all distinct.

Algorithm RandQuickSort(S)

Choose a pivot element x_i u.a.r from S

Split the set S into two subsets $S_1 = \{x_j | x_j < x_i\}$ and $S_2 = \{x_j | x_j > x_i\}$ by comparing each x_j with x_i

Recurse on sets S1 and S2

Output the sorted set S_1, x_i, and then sorted S_2.

end Algorithm

A Simple Example

Algorithm RandQuickSort(S)

Choose a pivot element x_i u.a.r from S

Split the set S into two subsets $S_1 = \{x_j | x_j < x_i\}$ and $S_2 = \{x_j | x_j > x_i\}$ by comparing each x_j with x_i

Recurse on sets S1 and S2

Output the sorted set S_1, x_i and then sorted S_2.

end Algorithm

- Let T(n) be the time taken by the procedure.
- What can we say about T(n)?

A Simple Example

Algorithm RandQuickSort(S)

Choose a pivot element x_i u.a.r from S

Split the set S into two subsets $S_1 = \{x_j | x_j < x_i\}$ and $S_2 = \{x_j | x_j > x_i\}$

Recurse on sets S1 and S2

Output the sorted set S_1, x_i and then sorted S_2.

end Algorithm

- What can we say about T(n)?
- The maximum value of T(n) occurs when the pivot element x_i is the largest/smallest element of the remaining set during each recursive call of the algorithm.
- In this case, $T(n) = n + (n - 1) + \cdots + 1 = O(n^2)$.
- This value of T(n) is reached with a very low probability of $1/n \cdot 1/(n-1) \cdot \cdots \cdot 1/2 \cdot 1 = 1/n!$.
- Also, the best case occurs when every pivot element splits the applicable set into two equal sized subsets and then $T(n) = O(n \ln n)$.

A Simple Example

Algorithm RandQuickSort(S)

Choose a pivot element x_i u.a.r from S

Split the set S into two subsets $S_1 = \{x_j | x_j < x_i\}$ and $S_2 = \{x_j | x_j > x_i\}$

Recurse on sets S1 and S2

Output the sorted set S_1, x_i and then sorted S_2.

end Algorithm

- This implies that T(n) has a distribution between O(n ln n) and $O(n^2)$.
- Now we derive the expected value of T(n).

A Simple Example

- This implies that T(n) has a distribution between O(n ln n) and $O(n^2)$.
- Now we derive the expected value of T(n).
- Note that if the i^{th} smallest element is chosen as the pivot element then S_1 and S_2 will be of sizes $i - 1$ and $n - i - 1$ respectively.
- And this choice has a probability of 1/n.
- Hence, the recurrence relation for T(n) is:
- $T(n) = n + T(X) + T(n - 1 - X)$
- In the above, X is a random variable indicating the size of S_1.

A Simple Example

- Note that if the ith smallest element is chosen as the pivot element then S1 and S2 will be of sizes i − 1 and n − i − 1 respectively.
- Hence, the recurrence relation for T(n) is:
- $T(n) = n + T(X) + T(n - 1 - X)$
- In the above, X is a random variable indicating the size of S_1.
- Further, note that Pr[X = i] = 1/n = Pr[n – 1 – X = i] as Pr[X = i] = 1/n.
- The last part is true since the choice of the pivot is uniform.

A Simple Example

- Hence, the recurrence relation for T(n) is:
- $T(n) = n + T(X) + T(n - 1 - X)$
- Further, note that Pr[X = i] = 1/n = Pr[n – 1 – X = i] as Pr[X = i] = 1/n.
- Taking expectations on both sides,
- $E[T(n)] = n + 1/n \sum_{i=1}^{n-1} E[T(i)] + (1/n) \sum_{i=1}^{n-1} E[T(i)]$.

- Use the fact that for a random variable Y with its support partitioned into sets $A_1, A_2, \ldots, A_n$, we have that $E[Y] = \sum_i Pr(A_i) . E[Y | A_i]$.

- Let f(i) = E[T(i)].

A Simple Example

- Taking expectations on both sides of,
- $E[T(n)] = n + 1/n \sum_{i=1}^{n-1} E[T(i)] + (1/n) \sum_{i=1}^{n-1} E[T(i)]$.

- Let $f(i) = E[T(i)]$.
- We can simplify the expression as $f(n) = n + (2/n) \sum_{i} f(i)$.
- Further simplification results in $nf(n) = n^2 + 2(f(1) + f(2) + ... + f(n - 1))$.

A Simple Example

- Further simplification results in

- $nf(n) = n^2 + 2(f(1) + f(2) + ... + f(n – 1)).$

- Write the above by replacing n with n – 1 to get

$(n – 1)\, f(n-1) = (n-1)^2 + 2(f(1) + f(2) + ... + f(n – 2)).$

- Subtract the two equation to get:

$nf(n) = n^2 + 2(f(1) + f(2) + ... + f(n – 1)).$

A Simple Example

- Further simplification results in
- $nf(n) = n^2 + 2(f(1) + f(2) + ... + f(n – 1)).$

- Write the above by replacing n with n – 1 to get
- $(n – 1) f(n-1) = (n-1)^2 + 2(f(1) + f(2) + ... + f(n – 2)).$

- Subtract the two equation to get:

  $nf(n) = n^2 + 2(f(1) + f(2) + ... + f(n – 1)).$

  or $f(n) = (n+1)/n\ f(n – 1) + (2n – 1)/n.$
- We prove by induction that $f(n) \leq 2n \ln n.$

A Simple Example

- f(n) = (n+1)/n f(n – 1) + (2n – 1)/n.
- We prove by induction that f(n) ≤ 2n ln n.
- Check the base case for n = 1.
- Let the result hold for all values of n up to n – 1.
- Induction step:

Let the claim hold for all values up to $n-1$. Then,

$$
\begin{aligned}
f(n) &= \frac{n+1}{n} f(n-1) + \frac{2n-1}{n} \\
&\leq \frac{n+1}{n} 2(n-1)\ln(n-1) + \frac{2n-1}{n} \text{ by induction hypothesis} \\
&= \frac{2(n^2-1)}{n} \ln(n-1) + \frac{2n-1}{n} \\
&= \frac{2(n^2-1)}{n} (\ln n + \ln(1-\frac{1}{n})) + \frac{2n-1}{n}
\end{aligned}
$$

We make use of the standard inequality stated below.

A Simple Example

Let the claim hold for all values up to $n-1$. Then,

$$\begin{aligned} f(n) &= \tfrac{n+1}{n} f(n-1) + \tfrac{2n-1}{n} \\ &\leq \tfrac{n+1}{n} 2(n-1)\ln(n-1) + \tfrac{2n-1}{n} \text{ by induction hypothesis} \\ &= \tfrac{2(n^2-1)}{n} \ln(n-1) + \tfrac{2n-1}{n} \\ &= \tfrac{2(n^2-1)}{n} (\ln n + \ln(1-\tfrac{1}{n})) + \tfrac{2n-1}{n} \end{aligned}$$

We make use of the standard inequality stated below.

$1 + x \leq e^x$ *for* $x \in R$.

Hence,

$$\begin{aligned} f(n) &\leq \tfrac{2(n^2-1)}{n} (\ln n - \tfrac{1}{n}) + \tfrac{2n-1}{n} \\ &= 2n \ln n - \tfrac{2}{n} \ln n - 2 + \tfrac{2}{n^2} + 2 - \tfrac{1}{n} \\ &\leq 2n \ln n, \text{ establishing the inductive step.} \end{aligned}$$

A Simple Example

- Hence, the expected running time of the randomized quick sort algorithm is O(n ln n).
- But one of the limitations of the recurrence relation approach is that we do not how the running time of the algorithm is spread around its expected value.
- Can this analysis be extended to answer questions such as, with what probability does the algorithm RandQuickSort needs more than 12n lnn time steps?
- Later on, we apply a different technique and establish that this probability is very small.
- To be able to answer such queries, we study Tail inequalities in the following.

Tail Inequalities

- We study three ways to estimate the tail probabilities of random variables.
- It will be noted that, the more information we know about the random variable the better the estimate we can derive about a given tail probability.

Tail Inequalities

- Markov Inequality: If X is a non-negative valued random variable with an expectation of μ, then $Pr[X \geq c\mu] \leq 1/c$.
- Proof of Markov inequality:

$$\mu = \sum_a a \cdot Pr(X = a)$$

$$= \sum_{a < c\mu} a \cdot Pr(X = a) + \sum_{a \geq c\mu} a \cdot Pr(X = a)$$

$$\geq 0 + \sum_{a \geq c\mu} a \cdot Pr(X = a)$$

$$\geq c\mu \cdot \sum_{a \geq c\mu} Pr(X = a)$$

$$= c\mu \cdot Pr(X \geq c\mu)$$

Tail Inequalities

- Markov Inequality: If X is a non-negative valued random variable with an expectation of μ, then $Pr[X \geq c\mu] \leq 1$.
- Applying this inequality tells us that the randomized quick sort algorithm has a run time of more than twice its expectation with a probability of 1/2.
- The run time is n^2 with probability of nearly log n /n.

Tail Inequalities

- Chebychev Inequality : We first define the terms standard deviation and variance of a random variable X.
- Let X be a random variable with an expectation of μ. The variance of X, denoted by var(X), is defined as $var(X) = E[(X - \mu)^2]$. The standard deviation of X, denoted by σ_X, is defined as $\sigma_X = var(X)$.
- Note that by definition, $var(x) = E[(X-\mu)^2] = E[X^2-2X\mu+\mu^2] = E[X^2]-\mu^2$.
- The second equality follows from the linearity of expectations.

Tail Inequalities

- Chebychev inequality: Let X be a random variable with expectation μX and standard deviation σX.
- Then, $Pr[|X - \mu_X| \geq c\sigma_X] \leq 1/c^2$

Tail Inequalities

- Chebychev inequality: Let X be a random variable with expectation μ_X and standard deviation σ_X.
- Then, $Pr[|X - \mu_X| \geq c\sigma_X] \leq 1$
- Proof. Let random variable $Y = (X - \mu_X)^2$. Then,

$$E[Y] = E[(X - \mu_X)^2] = \sigma_X^2 \text{ by definition}$$

$$\text{Now, } Pr[|X - \mu_X| \geq c\sigma_X] = Pr[(X - \mu_X)^2 \geq c^2\sigma_X^2]$$

$$= Pr[Y \geq c^2\sigma_X^2].$$

Applying Markov Inequality to the random variable Y

$$Pr[Y \geq c^2\sigma_X^2] = Pr[Y \geq c^2\mu_Y] \leq \frac{1}{c^2}$$

Tail Inequalities

- Better tail inequalitites can be obtained by the powerful technique called Chernoff bounds.
- However, applicability is a little restricted too.
- Let us study the most popular version to start with.

Tail Inequalities

- Let X be a random variable defined as the sum of n independent and identically distributed random variables $X_1, X_2, ..., X_n$.
  - In other words, $X = \Sigma_i X_i$
  - Short form i.i.d.
- Let us assume that each X_i is a Bernoulli random variable.
  - In other words, each X_i takes values in {0, 1}.
- Let $Pr(X_i = 1) = p$ and hence $Pr(X_i = 0) = 1 – p$.
- Finally, let $E[X] = \mu$.
  - Notice that $E[X] = \Sigma_i E[X_i] = n. (1.p + (1-p).0)) = np$.

Tail Inequalities

- Several settings relate to the above statements.
- Consider throwing a (biased) coin over n trials.
- Each trial, the probability of Heads is p.
- So, each X_i corresponds to the fact that the ith trial results in a Heads.
- Let us count the number of Heads over the n trails. Indeed, $X = \Sigma_i X_i$ captures this count as a random variable.
- Note that the expected number of Heads over n trails is exactly np.

Tail Inequalities

- Finally, to the theorem.
- Given the earlier conditions, it holds that for any $\delta > 0$,

$$Pr(X \geq \mu(1+\delta)) \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

- The normal strategy employed to prove tail estimates of sums of independent random variables is to make use of exponential moments.
- While proving Chebychev inequality, we made use of the second-order moment. It can be observed that using higher order moments would generally improve the bound on the tail inequality.
- But using exponential moments would result in a vast improvement.

Tail Inequalities

- Given the earlier conditions, it holds that for any $\delta > 0$,

$$Pr(X \geq \mu(1+\delta)) \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

- While proving Chebychev inequality, we made use of the second-order moment of a random variable.
- It can be observed that using higher order moments would generally improve the bound on the tail inequality
- But using exponential moments would result in a vast improvement in the bound.
- An exponential moment of a random variable X is the expectation of functions of X such as e^{X}.

Tail Inequalities

- Given the earlier conditions, it holds that for any $\delta > 0$,

$$\Pr(X \geq \mu(1+\delta)) \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

- For each i, we define the random variable $Y_i = e^{tX_i}$ for a real number $t > 0$ that will be chosen later.
- Notice that

1) Y_i is a positive valued random variable.

2) $E[Y_i] = E[e^{tX_i}] = p_i \cdot e^{t} + (1-p_i) \cdot e^{0}$

$$= p_i e^{t} + 1 - p_i$$

Tail Inequalities

- Given the earlier conditions, it holds that for any $\delta > 0$,

$$Pr(X \geq \mu(1+\delta)) \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

- Now define another random variable $Y = Y_1.Y_2....Y_n$.
- Now, we can note that

$$E[Y] = E[Y_1 \cdot Y_2 \cdots Y_n]$$

$$= \prod_{i=1}^{n} E[Y_i] = \left(p_i \cdot e^{t} + 1 - p_i\right)^{n}$$

- Why does the above calculation hold?

Tail Inequalities

- Given the earlier conditions, it holds that for any $\delta > 0$,

$$Pr(X \geq \mu(1+\delta)) \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

- The next step we do is do apply Markov inequality on Y as follows.
- First, notice that $Y = e^{tX}$ as

$$Y = Y_1 \cdot Y_2 \cdots Y_n = e^{tX_1} \cdot e^{tX_2} \cdots e^{tX_n}$$

$$= e^{t(X_1 + X_2 + \cdots + X_n)} = e^{tX}$$

Tail Inequalities

- Given the earlier conditions, it holds that for any $\delta > 0$,

$$\Pr(X \geq \mu(1+\delta)) \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

- First, notice that $Y = e^{tX}$. And,

- Further, $X \geq \mu(1+\delta) \Leftrightarrow e^{tX} \geq e^{t\mu(1+\delta)}$

$$\Rightarrow e^{Y} \geq e^{t\mu(1+\delta)}$$

- So, we are interested in the event $Y \geq e^{t\mu(1+\delta)}$.

Tail Inequalities

- Given the earlier conditions. it holds that for any $\delta > 0$,

$$\Pr(X \geq \mu(1+\delta)) \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

- So, we are interested in the event $Y \geq t\mu(1+\delta)$. We proceed as:

$$\Pr(Y \geq e^{t\mu(1+\delta)}) \leq \frac{E[Y]}{e^{t\mu(1+\delta)}} = \frac{\prod(1-p_i+p_i e^t)}{e^{t\mu(1+\delta)}}$$

$$\leq \frac{\prod e^{-p_i+p_i e^t}}{e^{t\mu(1+\delta)}}$$

$$= \frac{e^{-\mu(1-e^t)}}{e^{t\mu(1+\delta)}}$$

Tail Inequalities

- Given the earlier conditions, it holds that for any $\delta > 0$,

$$\Pr(X \geq \mu(1+\delta)) \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

- So, we are interested in the event $Y \geq t\mu(1+\delta)$. We proceed as:

$$\Pr(Y \geq e^{t\mu(1+\delta)}) \leq \frac{E[Y]}{e^{t\mu(1+\delta)}} = \frac{\prod(1-p_i+p_i e^t)}{e^{t\mu(1+\delta)}}$$

$$\leq \frac{\prod e^{-p_i+p_i e^t}}{e^{t\mu(1+\delta)}}$$

$$= \frac{e^{-\mu(1-e^t)}}{e^{t\mu(1+\delta)}} = e^{-\mu(1-e^t)-t\mu(1+\delta)}$$

Tail Inequalities

- Given the earlier conditions. it holds that for any $\delta > 0$,

$$\Pr(X \geq \mu(1+\delta)) \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

- So, $\Pr\left(Y \geq e^{t\mu(1+\delta)}\right) = e^{-\mu(1-e^{t}) - t\mu(1+\delta)}$
- Since t is a free parameter in the above, we can find a t that minimizes the right hand side.
- To simplify, let $f(t) = \ln e^{-\mu(1-e^{t}) - t\mu(1+\delta)}$

$$= -\mu(1-e^{t}) - t\mu(1+\delta)$$

Tail Inequalities

- Given the earlier conditions, it holds that for any $\delta > 0$,

$$Pr(X \geq \mu(1+\delta)) \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

- To simplify, let $f(t) = \ln e^{-\mu(1-e^t) - t\mu(1+\delta)}$

$$= -\mu(1-e^t) - t\mu(1+\delta)$$

- Differentiating f(t) wrt t, we get $f'(t) = \mu e^t - \mu(1+\delta)$

- So, f'(t) = 0 at $t = \ln(1+\delta)$

- Verify that the above t corresponds to a minima. OFFLINE.

Tail Inequalities

- Given the earlier conditions. it holds that for any $\delta > 0$,

$$\Pr(X \geq \mu(1+\delta)) \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

- With $t = \ln(1+\delta)$, we get that

$$\Pr(X \geq \mu(1+\delta)) \leq \frac{e^{-\mu(1-(1+\delta))}}{(1+\delta)^{\mu(1+\delta)}} = \frac{e^{\mu\delta}}{(1+\delta)^{1+\delta}}$$

$$= \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

- completing the proof.

Tail Inequalities

- Given the earlier conditions, it holds that for any $\delta > 0$,

$$\Pr(X \geq \mu(1+\delta)) \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

- With $t = \ln(1+\delta)$, we get that

$$\Pr(X \geq \mu(1+\delta)) \leq \frac{e^{-\mu(1-(1+\delta))}}{(1+\delta)^{\mu(1+\delta)}} = \frac{e^{\mu\delta}}{(1+\delta)^{1+\delta}}$$

$$= \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

- A simplification of the RHS gives

$$\Pr(X \geq \mu(1+\delta)) \leq \begin{cases} e^{-\mu\delta^2/4} & \text{if } \delta \leq 1 \\ e^{-\mu\delta\ln\delta} & \text{if } \delta > 1 \end{cases}$$

A Simple Example

- Here is one more practical application of the Chernoff bounds.
- Consider once again counting the number of heads out of tossing n fair coins independently.
  - So, p = ½.
- Let X_i denote the random variable that takes 1 if the ith coin toss results in a head, and 0 otherwise.
  - $E[X_i] = ½$.
- Let $X = \Sigma_i X_i$
- X counts the total number of heads over the n tosses.
  - E[X] = n/2.
  - With n = 100, say, we expect 50 heads over 100 coin tosses.

A Simple Example

- Markov inequality tells that the probability that X takes a value beyond 70 is Pr (X ≥ 70) = Pr (X ≥ 70/50 x 50) <= 5/7 = 0.7 (approx.)
- To apply Chbychev's inequality, we need to do some extra work.
- $Var(X_i)$ for any i is computed as $E[X_i^2] – E[X_i]^2$.
- $E [X_i^2] = 0x(1/2) + 1x(1/2) = ½$.
- $Var(X_i) = ½ - (½)^2 = ¼$.
- $Var(X) = 100x\ Var(X_1)$ due to independence.
- So, $Var(X) = 100/4 = 25$, and $\sigma_X = 5$.
- Now, Pr(X ≥ 70) can be rewritten as Pr(|X – 50| ≥ 20) and further as Pr(|X – 50| ≥ (20x $1/\sigma_X$) σ_X)) which is now at most 25/400 = 1/16 = 0.0625.

A Simple Example

- Markov inequality tells that the probability that X takes a value beyond 70 is Pr (X ≥ 70) = Pr (X ≥ 70/50 x 50) <= 5/7.
- Applying Chebychev's inequality, we get that Pr(X ≥ 70) is at most 1/16.
- This is quite an improvement on the upper bound of the probability of the event of interest.
- Let us see what Chernoff bounds will allow us to claim.
- We start by writing Pr(X ≥ 70) = Pr(X ≥ (1+2/5)x50)).
- From Chernoff bounds, this probability is at most $\exp\{20\}/(1.4)^{70}$ = 0.028 (approx.).

Applications of Tail Inequalities

- Here is one more practical application of the Chernoff bounds.
- Consider dividing a data set with features of interest into two parts: a test set and a training set.
- To make sure that both are roughly similar copies, you want to divide so that both data sets have the same number of data items of any given feature.
- Similar requirements arise in also experiments related to drug trails.

Example

Candidate	Above 50 Yrs	Diabetic	Frontline worker	Hypertension	Female
C1	1	0	1	1	0
C2	1	0	0	0	1
C3	0	0	0	0	1
C4	0	1	1	1	0
C5	1	1	0	0	0

Applications of Tail Inequalities

- To simplify matters, we will think of n data items with n features.
- Prepare a matrix A of nxn with entries from {0, 1}.
- Rows are features, columns are data items
- The goal is to find a vector x of size n with entries from {-1, +1} such that Ax has the smallest possible maximum absolute entry.
  - Rows with +1 in x belong to one class and those with -1 in x belong to another class.
  - The maximum absolute entry in Ax indicates how many data items differ at feature i according to the division by x.

Applications of Tail Inequalities

- Example

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \qquad x = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \qquad Ax = \begin{bmatrix} -2 \\ 1 \\ 0 \\ -2 \end{bmatrix}$$

- The maximum absolute entry in Ax is 2.

Applications of Tail Inequalities

- No good deterministic algorithms are known.
- The brute-force algorithm has to try all possible 2^n vectors.
- However, a very simple randomized algorithm exists.
- Consider choosing each element of x uniformly at random from {1, -1}.
- We will show that the maximum absolute entry of Ax in such an x is bounded by $O((n \ln n)^{1/2})$ with high probability.

Applications of Tail Inequalities

- Consider choosing each element of x uniformly at random from {1, -1}.
- We will show that the maximum absolute entry of AX in such an X is bounded by $O((n \ln n)^{1/2})$ with high probability.
- Let the product AX = Y.
- Consider any Y_i, say Y_1.
- By definition of matrix multiplication, $Y_1 = A_{11}X_1+A_{12}X_2+ \cdots +A_{1n}X_n$ where the A_{ij} denotes the element of A at the ith row and jth column.

Applications of Tail Inequalities

- Consider choosing each element of x uniformly at random from {1, -1}.
- We will show that the maximum absolute entry of AX in such an X is bounded by $O((n \ln n)^{1/2})$ with high probability.
- Let the product AX = Y.
- Consider any Y_i, say Y_1.
- By definition of matrix multiplication, $Y_1 = A_{11}X_1+A_{12}X_2+ \ldots +A_{1n}X_n$
- Note that $E[X_i] = 0$ and by linearity of expectations, $E[Y_1] = 0$.

Applications of Tail Inequalities

- We will show that the maximum absolute entry of AX in such an X is bounded by $O((n \ln n)^{1/2})$ with high probability.
- Let the product AX = Y. Consider any Y_i, say Y_1.
- By definition of matrix multiplication, $Y_1 = A_{11}X_1+A_{12}X_2+ \cdots +A_{1n}X_n$
- Note that $E[X_i] = 0$ and by linearity of expectations, $E[Y_1] = 0$.
- Let us imagine the case where the choices of X are made independently.
- We can then apply Chernoff bound on Y.

Applications of Tail Inequalities

- One small detour before we do that.
- In our Chernoff bound derived earlier, each X_i is {0,1} valued random variable.
- Now, each X_i is a +1/-1 valued random variable.
- Need a new version of Chernoff bound!

Applications of Tail Inequalities

- Consider X as the random variable that is the sum of n independent and identically distributed random variables X_i with X_i taking value among {-1, +1}.
- Let $Pr[X_i = +1] = Pr[X_i = -1] = 1/2$.
- Note that $E[X] = n.E[X_i] = n.0 = 0$.
- We want to know the Prob(X >= k) for some integral k.

Applications of Tail Inequalities

- Consider X as the random variable that is the sum of n independent and identically distributed random variables X_i with X_i taking value among {-1, +1}.
- Let $Pr[X_i = +1] = Pr[X_i = -1] = 1/2$.
- Note that $E[X] = n.E[X_i] = n.0 = 0$.
- We want to know the Prob(X >= k) for some integral k.
- Instead of doing the entire calculation again, let us do the following.
- Define $Y_i = (1+X_i)/2$. Now, Y_i is {0,1} valued.
- Define Y as the sum of Y_i's.

Applications of Tail Inequalities

- Consider X as the random variable that is the sum of n independent and identically distributed random variables X_i with X_i taking value among {-1, +1}.
- We want to know the Prob(X >= k) for some integral k.
- Define $Y_i = (1+X_i)/2$. Now, Yi is {0,1} valued.
- Define Y as the sum of Y_i's.
- Note that EY = n/2.
- Also, X >= k if and only if Y >= n/2 + k/2.
- Now, Pr(X >= k) = Pr(Y >= n/2 + k/2).

Applications of Tail Inequalities

- Also, X >= k if and only if Y >= n/2 + k/2.
- Now, Pr(X >= k) = Pr(Y >= n/2 + k/2).
- Rewrite as Pr(Y >= E[Y](1+(k/n)) with δ = k/n < 1.
- Applying Chernoff bounds with the above δ, we get that Pr(Y >= E[Y](1+d)) <= exp(-E[Y]δ^2/4) = exp(-δ^2/4n).

Applications of Tail Inequalities

- Back to our problem of set balancing....
- We now get $\Pr(Y_1 >= 8\sqrt{n \ln n}) \leq \exp(-64n \ln n) = \exp(-8 \ln n) = 1/n^8$.
- But we are interested in a two-sided bound.
- That is, since we want to minimize the absolute value of Y_1, we need to compute $\Pr[Y_1 \leq -d]$ also.
- But by symmetry, we have that $\Pr(Y_1 <= -d) <= 1/n^8$.
- So, $\Pr[|Y_1| >= (8n \ln n)^{1/2}] <= 2/n^8$.

Applications of Tail Inequalities

- Back to our problem of set balancing....
- So, $Pr[|Y_1| >= (8n \ln n)^{1/2}] <= 2/n^8$.
- But, what about Y_2, Y_3, etc.
- This is where another classical probability result aids us.
- Boole's inequality. For any events, E1, E2, ... , En,
- Pr(E1 U E2 U .. U En) <= Pr(E1) + Pr(E2) + ... + Pr(En).
- Apply the above to get that with probability at least $1-2/n^7$, every Y_i has an absolute value that is within $(8n \ln)^{1/2}$.

Applications of Randomization

- It is known that randomization combined with (non-trivial) algebraic techniques can lead to important applications.
- In this section, we showcase some of such techniques with respect to verification of identities.
- One such technique is called the fingerprinting technique described as follows.

Applications of Randomization

- Let U be any universe of objects and x and y be any two elements from U.
- The question we ask is, Is x = y?
- One can answer this using at least log |U| bits in a deterministic manner.
- However, consider mapping elements of U to a sparse universe V such that x and y are identical if and only if their images in V are identical, with a good chance.
- These images can be thought of as fingerprints of x and y.

Applications of Randomization

- Let us apply the above technique to matrix product verification. Let F be a field and A and B are two matrices with entries from F.
- Suppose it is claimed that $C = A \cdot B$.
- The fastest known matrix multiplication algorithm runs in time $O(n^{2.376})$.
- This algorithm is very difficult to implement, but the standard algorithms such as the Strassen's recursive algorithm takes time $O(n^{\log_2 7})$.
- So, to verify if C is indeed the product of A and B, it takes time equal to multiplying two matrices.

Applications of Randomization

- However, a simpler and efficient randomized approach exists.
- Let r be any vector with entires being 0 or 1.
- Let each element of r be chosen independently and uniformly at random.
- It is being assumed without loss of generality that 0 and 1 are the additive and multiplicative identities of the field F.

Applications of Randomization

- Compute $x = Br$, and $y = Ax$.
- Similarly, compute $z = Cr$.
- If $A \cdot B = C$ is indeed true, then y must equal z for any r.
- Also, x, y, and z can each be computed in $O(n^2)$ time.
- So, the time efficiency is established.
- It remains to see the verification efficiency.
- The following lemma argues that the verification procedure is efficient.

Applications of Randomization

- Lemma:1 Let A, B, and C be n × n matrices from F such that AxB ≠ C. Then, for r chosen uniformly at random from $\{0, 1\}^n$, Pr(ABr = Cr) ≤ 1/2.
- Proof. Consider the matrix D := AB − C. Since, AB ≠ C, matrix D is not the matrix of all zeros.
- We are interested in the event that Dr = 0.
- Assume without loss of generality that the first row of D has a nonzero entry and all all nonzero entries in that row are before any zero entry.

Applications of Randomization

- Lemma:1 Let A, B, and C be n × n matrices from F such that AxB ≠ C. Then, for r chosen uniformly at random from $\{0, 1\}^n$, Pr(ABr = Cr) ≤ 1/2.
- Proof.Consider the first row of D and the scalar obtained by multiplying the first row of D with r.
- The result is zero if and only if: $\mathbf{r}_1 = -\frac{\sum_{i=1}^{k} D_{1i}\mathbf{r}_i}{D_{11}}$
- In the above, it is assumed that there are k > 0 nonzero elements in the first row of D.

Applications of Randomization

- Lemma:1 Let A, B, and C be n × n matrices from F such that AxB ≠ C. Then, for r chosen uniformly at random from $\{0, 1\}^n$, Pr(ABr = Cr) ≤ 1/2.
- Proof. The above event is a super-event of the event that Dr = 0.
- Therefore the probability of the event Dr = 0 is upper bounded by the probability of the above event.
- To compute the above probability, imagine that all the choices $r_2, \cdots, r_k$ have been made.
- In that case, the right hand side is a scalar from the field F.
- The left hand size is a value uniformly chosen amongst (at least) two values in F. The required probability therefore cannot exceed 1/2.

Applications of Randomization

- To improve the verification efficiency of the procedure, we can also use repeated independent trials.
- Let us perform t independent trials of the above procedure.
- For $AB \neq C$, the probability that the test fails in each trial is at most 1/2.
- So, in t trials, the probability that all t trials fail, or a majority of the t trials fail, is at most $(1/2)^t$.
- For $t = O(\log n)$, the failure probability is polynomially small

Applications of Randomization

- Other applications of the technique include verifying polynomial identities.
- For instance, let $P_1(x)$, $P_2(x)$ be two polynomials in a field F.
- The polynomial product verification problem is to check whether $P_1(x) \cdot P_2(x) = P_3(x)$ for a given $P_3(x)$.
- It holds that there exists an O(n log n) time algorithm to multiply two polynomials, where n is themaximum degree of P_1 and P_2.
- We design a verification algorithm that is faster than O(n log n).

Applications of Randomization

- Let $S \subset F$ be a susbet of size at least $2n + 1$.
- The main idea of the verification procedure is that if indeed $P_3(x)$ equals $P_1(x) \cdot P_2(x)$, then, also $P_3(r) = P_1(r) \cdot P_2(r)$ for r chosen uniformly at random from S.
- Further, evaluating a polynomial at a given input can be done in O(n) time.
- So, we can declare that $P_3(x)$ equals $P_1(x) \cdot P_2(x)$ unless $P_3(r) \neq P_1(r) \cdot P_2(r)$.
- The algorithm makes a mistake only when indeed $P_3(x) \neq P_1(x) \cdot P_2(x)$ but the choice of r fails to detect this.

Applications of Randomization

- To estimate the probability that the algorithm makes a mistake, let $Q(x) := P_3(x) - P_1(x) \cdot P_2(x)$.
- The degree of Q(x) is at most 2n.
- Suppose that $P_3(x) \neq P_1(x) \cdot P_2(x)$.
- Then, Q(x) is a nonzero polynomial.
- So the test fails if Q(r) = 0.
- However, the polynomial Q(x) of degree at most 2n can have at most 2n roots.
- So, the probability that Q(r) = 0 is at most 2n/|S|, which is also the probability of error.
- As earlier, the probability of failure can be made polynomially small in n by using repeated trails or choosing a larger S, or both.

Applications of Randomization

- One may wonder whether it is at all worthwhile to have elaborate verification algorithms for things as simple as polynomial product verification.
- Such techniques however are more applicable when polynomials are not available explicitly.

Finger-Printing

- The two algorithms that we considered today have the property that for inputs that are identical, the algorithm does not make any error.
- But in inputs that are not identical, the algorithm makes an error that is upper bounded by at least a constant.
- Repeated runs of the same algorithm can catch the error, hence the error can be made arbitrarily small.

Finger-Printing

- On inputs that are not identical, the algorithm makes an error that is upper bounded by at least a constant.
- Repeated runs of the same algorithm can catch the error, hence the error can be made arbitrarily small.
- Consider A to be a finger-printing algorithm.
- Let us run A on input x, y for t iterations.
- The t outputs are, say, o_1, o_2, ..., o_t.
- If any of these t outputs are NO, then we can return NO as the answer.
- If all t are YES, then we return YES as the answer.
- Given that x = y,

Finger-Printing

- Repeated runs of the same algorithm can catch the error, hence the error can be made arbitrarily small.
- Consider A to be a finger-printing algorithm.
- Let us run A on input x, y for t iterations.
- The t outputs are, say, o_1, o_2, ..., o_t.
- If any of these t outputs are NO, then we can return NO as the answer.
- If all t are YES, then we return YES as the answer.
- Given that $x = y$, $Pr(A(x,y) = YES) = 1$.
- Given that $x \neq y$, $Pr(A(x,y) = YES) \leq (1/2)^t$.
- So, if $t = O(\log n)$, then the error probability is $O(1/n^c)$.

RP and co-RP

- The above algorithms are called as co-RP algorithms.
- RP stands for Randomized Polynomial.
- Definition: The class RP consists of languages L such that there exists a randomized algorithm running in worst case polynomial time such that for any input x:
  - $x \in L \Rightarrow$ Pr(A accepts x) ≥ 1/2.
  - $x \notin L \Rightarrow$ Pr(A accepts x) = 0

- The complement of the class RP is the class co-RP.
- Note that any RP or co-RP algorithm can err only on one side, either for x in L, or for x not in L.

RP and co-RP

- In definition: The class RP consists of languages L such that there exists a randomized algorithm running in worst case polynomial time such that for any input x:
  - $x \in L \Rightarrow$ Pr(A accepts x) $\geq 1/2$.
  - $x \notin L \Rightarrow$ Pr(A accepts x) = 0

- The complement of the class RP is the class co-RP.
- In definition: The class co-RP consists of languages L such that there exists a randomized algorithm running in worst case polynomial time such that for any input x:
  - $x \in L \Rightarrow$ Pr(A accepts x) = 0.
  - $x \notin L \Rightarrow$ Pr(A accepts x) $\leq 1/2$.

RP and co-RP

- In definition: The class RP consists of languages L such that there exists a randomized algorithm running in worst case polynomial time such that for any input x:
  - $x \in L \Rightarrow$ Pr(A accepts x) $\geq 1/2$.
  - $x \notin L \Rightarrow$ Pr(A accepts x) = 0
- In definition: The class co-RP consists of languages L such that there exists a randomized algorithm running in worst case polynomial time such that for any input x:
  - $x \in L \Rightarrow$ Pr(A accepts x) = 0.
  - $x \notin L \Rightarrow$ Pr(A accepts x) $\leq 1/2$
- Note that any RP or co-RP algorithm can err only on one side, either for x in L, or for x not in L.

RP and co-RP

- In definition: The class RP consists of languages L such that there exists a randomized algorithm running in worst case polynomial time such that for any input x:
  - $x \in L \Rightarrow$ Pr(A accepts x) ≥ 1/2.
  - $x \notin L \Rightarrow$ Pr(A accepts x) = 0
- Can you recall and RP algorithms that we studied?
-

RP and co-RP

- In definition: The class RP consists of languages L such that there exists a randomized algorithm running in worst case polynomial time such that for any input x:
  - $x \in L \Rightarrow \Pr(A \text{ accepts } x) \geq 1/2$.
  - $x \notin L \Rightarrow \Pr(A \text{ accepts } x) = 0$

- Can you recall and RP algorithms that we studied?
- The interactive proof algorithm for graph non-isomorphism is an example.

What about Randomized QuickSort

- The randomized quick sort algorithm does not make any errors in its output.
- So, clearly, it does not fit into either of RP or co-RP.
- While there are no errors in its output, we recall that its run time may vary.
- Such algorithms are called ZP algorithms.
  - Stands for Zero Error Expected Polynomial, time
- The class ZP consists of languages L such that there is a randomized algorithm A that always outputs the correct answer while running in expected polynomial time.
- Another name for ZP algorithms is Las Vegas algorithms.

Proof by Existence

- Many times you want to show that a particular combinatorial object exists.
- May be very inefficient to build possibly because of a huge space and a small target of interest.
  - Like finding a needle in a haystack.
- This is where randomization can come to the resuce.

Proof by Existence

- Two useful statements:

1) If a random variable has expected value $E[X] = a$, then certainly there exists a realisation of X with value $\geq a$ and a realisation of X with value $\leq a$.

2) If a random object drawn from some universe of objects has a certain property with non-zero probability then there must exist an object with that property in this universe.

Proof by Existence

- Two useful statements:

1) If a random variable has expected value E[X] = a, then certainly there exists a realisation of X with value ≥ a and a realisation of X with value ≤ a.

2) If a random object drawn from some universe of objects has a certain property with non-zero probability then there must exist an object with that property in this universe.

- Each of these statements looks simple on their own, but they are remarkably powerful in Computer Science.

Proof by Existence

- We will start with a simple example.
- Consider an undirected graph G = (V, E).
- We want to find a subgraph G' of G that:

  1) has the largest number of edges of G, and

  2) G' is bipartite.
- We will first show the existence of a G' with |E(G')| at least |E(G)|/2.

Proof by Existence

- Consider an undirected graph G = (V, E).
- We want to find a subgraph G' of G that:

  1) has the largest number of edges of G, and

  2) G' is bipartite.
- We will first show the existence of a G' with |E(G')| at least |E(G)|/2.
- The random experiment we perform is to assign a bit 0 or 1, denoted b(v), to each vertex v of G ind. and uar.
- Put all vertices in G'.
- An edge uv is in G' if and only if b(u) is different from b(v).

Proof by Existence

- The random experiment we perform is to assign a bit 0 or 1, denoted b(v), to each vertex v of G ind. and uar.
- Put all vertices in G'.
- An edge uv is in G' if and only if b(u) is different from b(v).
- In other words, G' = (V_0 U V_1, V_0xV_1 int. E(G)).
- Let us now bound |E(G')|.
- Let X_{uv} be a random variable that indicates the event {uv in E(G')}.
- $E[X_{uv}]$ = Pr({uv in E(G')}) = 1/2.

Proof by Existence

- The random experiment we perform is to assign a bit 0 or 1, denoted b(v), to each vertex v of G ind. and uar.
- An edge uv is in G' if and only if b(u) is different from b(v).
- Let us now bound |E(G')|.
- Let X_{uv} be a random variable that indicates the event {uv in E(G')}.
- $E[X_{uv}] = Pr(\{uv \text{ in } E(G')\}) = 1/2$.
- Let $X = \sum_{uv \text{ in } E(G)} X_{uv}$.
- $E[X] = E[\sum_{uv \text{ in } E(G)} X_{uv}] = \sum_{uv \text{ in } E(G)} E[X_{uv}] = |E(G)|/2$.

Proof by Existence

- The random experiment we perform is to assign a bit 0 or 1, denoted b(v), to each vertex v of G ind. and uar.
- Let X_{uv} be a random variable that indicates the event {uv in E(G')}.
- $E[X_{uv}] = Pr(\{uv \text{ in } E(G')\}) = 1/2$.
- Let $X = \sum_{uv \text{ in } E(G)} X_{uv}$.
- $E[X] = E[\sum_{uv \text{ in } E(G)} X_{uv}] = \sum_{uv \text{ in } E(G)} E[X_{uv}] = |E(G)|/2$.
- By Statement (1) earlier, there must exist an assignment of b() to vertices such that G' has at least half the edges of G.

Proof by Existence

- Let us now move to a more involved example.
- We start with the definition of an (α, β, n, d) expander.
- A bipartite graph G = (V_1 U V_2, E) on n nodes is an (α, β, n, d) expander if

1) Every vertex in V1 has degree at most d.

2) For any subset S of vertices from V_1 such that |S| is at most αn, there are at least β|S| neighbors in V_2.

- Ideally, d should be small and β as large as possible.

Proof by Existence

- Let us now move to a more involved example.
- We start with the definition of an (α, β, n, d) expander.
- A bipartite graph G = (V_1 U V_2, E) on n nodes is an (α, β, n, d) expander if

1) Every vertex in V1 has degree at most d.

2) For any subset S of vertices from V_1 such that |S| is at most αn, there are at least β|S| neighbors in V_2.

- To build such a graph in a deterministic manner is not easy.
- Simple randomized construction with d = 18, α = 1/3, and β = 2

Proof by Existence

- A bipartite graph G = (V_1 U V_2, E) on n nodes is an (α, β, n, d) expander if

1) Every vertex in V1 has degree at most d.

2) For any subset S of vertices from V_1 such that |S| is at most αn, there are at least β|S| neighbors in V_2.

- Simple randomized construction with d = 18, α = 1/3, and β = 2.
- We will actually not use these values until the very end of the proof.

Proof by Existence

- A bipartite graph G = (V_1 U V_2, E) on n nodes is an (α, β, n, d) expander if

1) Every vertex in V1 has degree at most d.

2) For any subset S of vertices from V_1 such that |S| is at most αn, there are at least β|S| neighbors in V_2.

- Let each vertex v in V1 choose d neighbors in V2 by sampling independently and uniformly at random.
- We can even sample with replacement.
- In other words, the same choice can be made more than once.
- We will still consider only one copy of any multiple choices.

Proof by Existence

- A bipartite graph G = (V_1 U V_2, E) on n nodes is an (α, β, n, d) expander if

1) Every vertex in V1 has degree at most d.

- Let each vertex v in V_1 choose d neighbors in V_2 by sampling independently and uniformly at random.
- By this construction, each vertex in V1 has degree at most d.
- Next, we show the second condition.

Proof by Existence

- Condition 2: For any subset S of vertices from V_1 such that |S| is at most αn, there are at least $\beta|S|$ neighbors in V_2.
- Let $|V_1| = |V_2| = n$.
- Let each vertex v in V_1 choose d neighbors in V_2 by sampling independently and uniformly at random.
- Consider any subset S of V_1 with $|S| = s$.
- Let T be any subset of V_2 of size $< \beta s$.
- Consider the event that all the neighbors of vertices in S are in T.
- This event occurs with probability $(\beta s/n)^{ds}$.

Proof by Existence

- Condition 2: For any subset S of vertices from V_1 such that |S| is at most αn, there are at least $\beta|S|$ neighbors in V_2.
- Let $|V_1| = |V_2| = n$. Let each vertex v in V_1 choose d neighbors in V_2 by sampling independently and uar.
- Consider any subset S of V_1 with |S| = s.
- Let T be any subset of V_2 of size < βs.
- Consider the event that all the neighbors of vertices in S are in T.
- This event occurs with probability at most $(\beta s/n)^{ds}$.
- We have to now look at all possible S and all possible T.

Proof by Existence

- Condition 2: For any subset S of vertices from V_1 such that |S| is at most αn, there are at least $\beta|S|$ neighbors in V_2.
- Consider the event that all the ds neighbors of vertices in S of size s are in T.
- This event occurs with probability at most $(\beta s/n)^{ds}$.
- Let us use Boole's inequality to upper bound the probability of the event that for some S all its neighbors are in T.
- We have to now look at all possible S and all possible T.
- There are ${}^{n}C_{s}$ ways to choose S and ${}^{n}C_{\beta s}$ ways to choose T.

Proof by Existence

- Condition 2: For any subset S of vertices from V_1 such that |S| is at most αn, there are at least $\beta|S|$ neighbors in V_2.
- This event occurs with probability at most $(\beta s/n)^{ds}$.
- There are ${}^{n}C_{s}$ ways to choose S and ${}^{n}C_{\beta s}$ ways to choose T.
- The probability that for some S all its neighbors are in T is now upper bounded by ${}^{n}C_{s} \cdot {}^{n}C_{\beta s} \cdot (\beta s/n)^{ds}$.
- To simplify, let us use the inequality that for any n and k, ${}^{n}C_{k}$ is at most $(en/k)^{k}$.

Proof by Existence

- Condition 2: For any subset S of vertices from V_1 such that |S| is at most αn, there are at least $\beta|S|$ neighbors in V_2.
- The probability that for some S all its neighbors are in T is now upper bounded by ${}^{n}C_{s} \cdot {}^{n}C_{\beta s} \cdot (\beta s/n)^{ds}$.
- To simplify, let us use the inequality that for any n and k, ${}^{n}C_{k}$ is at most $(en/k)^{k}$.
- The probability is at most $(en/s)^{s} \cdot (en/\beta s)^{\beta s} \cdot (\beta s/n)^{ds}$.
- Simplifying we get, $\left[(s/n)^{d-\beta-1} \; e^{1+\beta} \; \beta^{d-\beta}\right]^{s}$.
- Use that s is at most αn, for $\alpha = 1/3$ to simplify to:

Proof by Existence

- Condition 2: For any subset S of vertices from V_1 such that |S| is at most αn, there are at least $\beta|S|$ neighbors in V_2.
- The probability that for some S all its neighbors are in T is now upper bounded by ${}^{n}C_{s} \cdot {}^{n}C_{bs} \cdot (\beta s/n)^{ds}$.
- The probability is at most $(en/s)^{s} \cdot (en/\beta s)^{\beta s} \cdot (\beta s/n)^{ds}$.
- Simplifying we get, $\left[(s/n)^{d-\beta-1} \; e^{1+\beta} \; \beta^{d-\beta}\right]^{s}$.
- Use that s is at most αn, for $\alpha = 1/3$ to simplify the above to $\left[(\beta/3)^{d} \; (3e)^{1+\beta}\right]^{s}$.
- Use d = 18 and $\beta = 2$ to simplify to $\left[(2/3)^{18} \; (3e)^{3}\right]^{s}$.

Proof by Existence

- Condition 2: For any subset S of vertices from V_1 such that |S| is at most αn, there are at least $\beta|S|$ neighbors in V_2.
- The probability that for some S all its neighbors are in T is at most $(en/s)^s \cdot (en/\beta s)^{\beta s} \cdot (\beta s/n)^{ds}$.
- Use that s is at most αn, for $\alpha = 1/3$ to simplify the above to $\left[(\beta/3)^d \ (3e)^{1+\beta}\right]^s$.
- Use d = 18 and $\beta = 2$ to simplify to $\left[(2/3)^{18} \ (3e)^3\right]^s$.
- Notice that the term in [ ] is at most 1/2. So, the entire probability is at most $(1/2)^s$.

Proof by Existence

- Condition 2: For any subset S of vertices from V_1 such that |S| is at most αn, there are at least $\beta|S|$ neighbors in V_2.
- The probability that for some S all its neighbors are in T is at most $\left[(2/3)^{18} \ (3e)^3\right]^s$.
- Notice that the term in [ ] is at most 1/2. So, the entire probability is at most $(1/2)^s$.
- We used a specific s. But, we need to show the result for all s between 1 to αn.
- Apply Boole's inequality again to get that

$$\sum_{s>0} \Pr(\text{for some S all its neighbors are in T})$$
$$\leq \sum_{s>0} (1/2)^s < 1$$

Proof by Existence--AnotherExample

- Consider the following claim.
- There is a bipartite graph G = (L, R, E) such that
  - $|L| = n$
  - $|R| = 2^{\log^2 n}$
  - Every subset of n/2 vertices of L has at least $2^{\log^2 n} - n$ neighbors in R.
  - No vertex of R has more than $12\log^2 n$ neighbors.
- We want to use the technique of proof by existence to show the above claim.

Proof by Existence--AnotherExample

- There is a bipartite graph G = (L, R, E) such that
  - $|L| = n$, $|R| = 2^{\log^2 n}$, . Every subset of n/2 vertices of L has at least $2^{\log^2 n} - n$ neighbors in R. No vertex of R has more than $12\log^2 n$ neighbors.
  - Let every vertex of L choose d neighbors in R independently and uniformly at random.
  - Choices are made with replacement.
  - Multiple edges are dropped in favor of one edge.

Proof by Existence--AnotherExample

- There is a bipartite graph G = (L, R, E) such that
  - $|L| = n$, $|R| = 2^{\log^2 n}$. Every subset of n/2 vertices of L has at least $2^{\log^2 n} - n$ neighbors in R. No vertex of R has more than $12\log^2 n$ neighbors.
  - Let every vertex of L choose d neighbors in R independently and uniformly at random.
  - Let us now estimate the degree of any vertex of R.
  - Let $|R| = r$.
  - We can think of the degree of a vertex v in R as the expectation of the random variable X that indicates how many vertices in L choose v as a neighbor.
  - Each neighbor in L makes d choices, so we have nd choices in all.

Proof by Existence--AnotherExample

- There is a bipartite graph G = (L, R, E) such that
  - $|L| = n$, $|R| = 2^{\log^2 n}$. Every subset of n/2 vertices of L has at least $2^{\log^2 n} - n$ neighbors in R. No vertex of R has more than $12\log^2 n$ neighbors.
  - Let every vertex of L choose d neighbors in R independently and uniformly at random.
  - Let $|R| = r$.
  - We can think of the degree of a vertex v in R as the expectation of the random variable X that indicates how many vertices in L choose v as a neighbor.
  - Each neighbor in L makes d choices, so we have nd choices in all.
  - Let Xi be a random variable if the ith choice is v.

Proof by Existence--AnotherExample

- Let |R| = r.
- We can think of the degree of a vertex v in R as the expectation of the random variable X that indicates how many vertices in L choose v as a neighbor.
- Each neighbor in L makes d choices, so we have nd choices in all.
- Let Xi be a random variable if the ith choice is v.
- E[Xi] = 1/r.
- $X = \Sigma$ Xi and so E[X] = Σ E[Xi] = nd/r.
- Pick d = r.2log^2 n /n so that E[X] = 2log^2 n.
- Now apply Chernoff bounds on X for the event X >= 12log^{2}n.
- Use Boole's inequality to bound the probability of the bad event for every v in R.

Proof by Existence--AnotherExample

- There is a bipartite graph G = (L, R, E) such that
  - $|L| = n$, $|R| = 2^{\log^2 n}$, . Every subset of n/2 vertices of L has at least $2^{\log^2 n} - n$ neighbors in R.
  - Let every vertex of L choose d neighbors in R independently and uniformly at random.
  - We now move to property 1.
  - Let S be any subset of size n/2 from L.
  - Let T be any subset of R of size $2^{\log^2 n} - n$.
  - Consider the event that all the neighbors of S are in T.
  - This happens with a probability of $\left[(2^{\log^2 n} - n)/n\right]^{nd/2}$.

Proof by Existence--AnotherExample

- There is a bipartite graph G = (L, R, E) such that
  - $|L| = n$, $|R| = 2^{\log^2 n}$. Every subset of n/2 vertices of L has at least $2^{\log^2 n} - n$ neighbors in R.
  - Let S be any subset of size n/2 from L.
  - Let T be any subset of R of size $2^{\log^2 n} - n$.
  - Consider the event that all the neighbors of S are in T.
  - This happens with a probability of $[(r - n)/r]^{nd/2}$.
  - Now, consider all possible choices of S and T. The probability that for any S all its neighbors are in some T is upper bounded by: ${}^{n}C_{n/2} \cdot {}^{r}C_{r-n} \cdot [(r - n)/r]^{nd/2}$.
  - We will now show that the above probability is at most 1.

Proof by Existence--AnotherExample

- Now, consider all possible choices of S and T. The probability that for any S all its neighbors are in some T is upper bounded by: ${}^{n}C_{n/2} \cdot {}^{r}C_{r-n} \cdot [(r-n)/r]^{nd/2}$.
  - We will now show that the above probability is at most 1.
  - Use the (in)equalities
    - ${}^{n}C_{n-k} = {}^{n}C_{k}$ for k between 0 and n.
    - ${}^{N}c_{k}$ is at most $(en/k)^{k}$.
    - (1+x) is at most e^{x} for any real number x.
  - The required probability is
  - $(2e)^{n/2} . (er/n)^{n} . (e)^{-n^{2}d/2r}$.
  - Recall that $d = 2\log^{2}n \cdot r/n$.

Proof by Existence--AnotherExample

- Now, consider all possible choices of S and T. The probability that for any S all its neighbors are in some T is upper bounded by: ${}^{n}C_{n/2} \cdot {}^{r}C_{r-n} \cdot [(r-n)/r]^{nd/2}$.
  - We will now show that the above probability is at most 1.
  - Use the (in)equalities
    - ${}^{n}C_{n-k} = {}^{n}C_{k}$ for k between 0 and n.
    - ${}^{N}c_{k}$ is at most $(en/k)^{k}$.
    - (1+x) is at most e^{x} for any real number x.
  - The required probability is
  - $(2e)^{n/2} . (er/n)^{n} . (e)^{-n^{2}d/2r}$.
  - Recall that $d = 2\log^{2}n \cdot r/n$ and $\log r = \log^{2}n$ to simplify.

Yet Another Example

- Consider a Boolean formula in CNF.
- In CNF, each clause is a disjunction of literals
- The formula is a conjunction of clauses.
- Another name for CNF is Product-of-Sums.

Yet Another Example

- We show that for any set of m clauses, there is a truth assignment that satisfies at least m/2 clauses.
- Proof: Consider a random assignment of truth values to variables as T/F.
- Consider a clause C_i of k variables.
- C_i is not satisfied with probability 2^{-k}.
- Define a random variable Zi that indicates the event Ci is satisfied.
- E[Zi] = Pr(Ci is satisfied) = $1 - 2^{-k}$.
- Define Z as the number of clauses satisfied. $Z = \sum Zi$.
- $E[Z] = E[\sum Zi] = \sum E[Zi] = m(1 - 2^{-k}) >= m/2$ as $k >= 1$.

Yet Another Example

- We show that for any set of m clauses, there is a truth assignment that satisfies at least m/2 clauses.
- Proof: Consider a random assignment of truth values to variables as T/F.
- $E[Z] = E[\sum Zi] = \sum E[Zi] = m(1 - 2^{-k}) >= m/2$ as $k >= 1$.
- The above holds irrespective of whether the formula is satisfiable or not.
- This version of the problem where we intend to maximize the number of clauses that can be satisfied is called as MAXSAT.
- MAXSAT is also NP-hard indicating that no good polynomial solutions exist.

Yet Another Example

- This version of the problem where we intend to maximize the number of clauses that can be satisfied is called as MAXSAT.
- Define for an instance I, m*(I) to be the maximum number of clauses that can be satisfied.
- Let $m^A(I)$ be the number of (expected) clauses that can be satisfied by an (randomized) algorithm A.
- The ratio $m^A(I)/m^*(I)$ is the performance ratio of algorithm A.
- We seek algorithms that this ratio as close to 1.
- The previous approach gives us 1/2 as the ratio.

Yet Another Example

- This version of the problem where we intend to maximize the number of clauses that can be satisfied is called as MAXSAT.
- The ratio $m^A(I)/m^*(I)$ is the performance ratio of algorithm A.
- We seek algorithms that this ratio as close to 1.
- The previous approach gives us 1/2 as the ratio.
  - Actually the ratio is $1-2^{-k}$.
- In fact, there are instances where one can satisfy only 1/2 of the clauses.

LP Rounding

- This version of the problem where we intend to maximize the number of clauses that can be satisfied is called as MAXSAT.
- We now study an approach that does better than 1/2.
- Finally, we devise an algorithm that gets us a ratio of 3/4.

LP Rounding

- The technique of LP Rounding uses the following approach.
- Write the optimization problem as an integer linear program (ILP).
- Relax some of the constraints of the ILP in a step called LP Relaxation to conver the ILP to a simpe Linear Program (LP).
- Note that LP can be solved in polynomial time. Get an optimal solution to the LP.
- Round the solution from LP to satisfy the integrality constraints.
- May lose some quality in this step but that is inevitable.

LP Rounding

- Let us apply LP rounding to the MAXSAT problem.
- Consider a clause C_i.
- An indicator variable z_i with values in {0, 1} is defined to indicate whether C_i is satisfied or not.
- We now seek to maximize $\sum_i z_i$.
- For each variable x_j, we define an indicator variable y_j that takes values 1 or 0 corresponding to x_j = True or False respectively.
- Since variables can appear in either the pure form or the complemented form, we separate these as follows.

LP Rounding

- Let us apply LP rounding to the MAXSAT problem.
- Consider a clause C_i.
- For each variable x_j, we define an indicator variable yj that takes values 1 or 0 corresponding to x_j = True or False respectively.
- Since variables can appear in either the pure form or the complemented form, we separate these as follows.
- Define C_{i+} to be the indices of variables that appear in pure form in C_i.
- Define C_{i-} to be the indices of variables that appear in pure form in C_i.

LP Rounding

- Let us apply LP rounding to the MAXSAT problem.
- Consider a clause C_i.
- For each variable x_j, we define an indicator variable y_j that takes values 1 or 0 corresponding to x_j = True or False respectively.
- Define C_{i+} to be the indices of variables that appear in pure form in C_i.
- Define C_{i-} to be the indices of variables that appear in pure form in C_i.
- Now, clause C_i is satisfied if it holds that for each i

$$\sum_{j \text{ in } C_{i+}} y_j + \sum_{j \text{ in } C_{i-}} (1 - y_j) >= z_i.$$

LP Rounding

- Let us apply LP rounding to the MAXSAT problem.
- The entire integer linear program is:

Maximize $\sum_i z_i$

subject to

$$\sum_{j \text{ in } C_{i+}} y_j + \sum_{j \text{ in } C_{i-}} (1 - y_j) >= z_i \text{ for all i.}$$

where y_j, z_i in {0, 1} for all i and j.

LP Rounding

- Let us apply LP rounding to the MAXSAT problem.
- Let us relax the constraints on y_j and z_i so that they can take values in [0,1]
- Note they are real numbers between 0 and 1 now and not just integral necessarily.
- We will use u_j and v_i for the values of the best solution to the relaxed linear program.
  - We use u's for the variables and v's for the clauses.
- Notice that $\sum_i v_i$ is an upper bound on the number of clauses that can be satisfied.
- But, the values of u_j are not integral, so they do not yet correspond to True/False values in a truth assignment.

LP Rounding

- Let us relax the constraints on y_j and z_i so that they can take values in [0,1]
- We will use u_j and v_i for the values of the best solution to the relaxed linear program.
- But, the values of u_j are not integral, so they do not yet correspond to True/False values in a truth assignment.
- The next step in the technique suggests to round the u_i's so that a truth assignment can be obtained. This step is called randomized rounding.
- Our rounding does the following: Set y_j to 1 with probability u_j.
  - This sets x_j to True with the same probability.

LP Rounding

- The next step in the technique suggests to round the u_j's so that a truth assignment can be obtained. This step is called randomized rounding.
- Our rounding does the following: Set y_j to 1 with probability u_j.
  - This sets x_j to True with the same probability.
- We now estimate the probability that a clause C_i is satisfied.

LP Rounding

- We now estimate the probability that a clause C_i is satisfied.
- Claim: A clause C_i with k literals is satisfied with probability at least $1 – (1-1/k)^k.v_i$.
- Recall what is v_i.
- Let us assume wlog that all the variables in C_i appear in their pure form.
- So, $C_i = x_1 \vee x_2 \vee \ldots x_k$ for some variables x_1 through x_k.
- In the relaxed LP, we satisfy the constraint $u_1 + u_2 + \ldots + u_k >= v_i$.
- C_i now remains unsatisfied if the corresponding x_1 through x_k are all 0.

LP Rounding

- A clause C_i with k literals is satisfied with probability at least $1 – (1-1/k)^k.v_i$.
- Recall what is v_i.
- So, $C_i = x_1 \vee x_2 \vee \ldots x_k$ for some variables x_1 through x_k.
- In the relaxed LP, we satisfy the constraint $u_1 + u_2 + \ldots + u_k >= v_i$.
- C_i now remains unsatisfied if the corresponding x_1 through x_k are all 0.
- This happens with probability $(1-u_j)$ for each variable and hence with probability $\prod_j (1-u_j)$ for the k variables.
- So, C_i is satisfied with probability $1 – \prod_j (1-u_j)$.

LP Rounding

- A clause C_i with k literals is satisfied with probability at least $1 – (1\text{-}1/k)^k.v_i$.
- This happens with probability $(1\text{-}u_j)$ for each variable and hence with probability $\prod_j (1\text{-}u_j)$ for the k variables.
- So, C_i is satisfied with probability $1 – \prod_j (1\text{-}u_j)$.
- We claim that the above is minimized when $u_j = v_i/k$ for each j. (Take the proof as a reading exercise).
- So, the probability of interest is $1 – (1 – v_i/k)^k$.
- We now claim that the function $f(r) = 1 – (1 – r/k)^k$ is at least $1 – (1\text{-}1/k)^k.r$ for all r in [0,1].
  - Take the proof of the above also as a reading exercise. You need to show that the function is concave.

LP Rounding

- A clause C_i with k literals is satisfied with probability at least $1 – (1\text{-}1/k)^k.v_i$.
- So, the probability of interest is $1 – (1 – v_i/k)^k$.
- We now claim that the function $f(r) = 1 – (1 – r/k)^k$ is at least $1 – (1\text{-}1/k)^k.r$ for all r in [0,1].
- By the above, we conclude that C_i is satisfied with probability at least $(1\text{-}1/k)^k.v_i$.
- Now, use linearity of expectations (over clauses) to show that the expected number of satisfied clauses is at least

$$\sum_i (1 – (1\text{-}1/k)^k).v_i >= (1 – 1/e) \sum_i v_i >= (1\text{-}1/e)\ m^*(I).$$

LP Rounding

- There are two algorithms with performance guarantees as shown:

k	$1 – (1/2^k)$	$1 – (1 – 1/k)^k$
1	0.5	1.0
2	0.75	0.75
3	0.875	0.703
4	0.938	0.684
5	0.969	0.672

- As k, the length of the clause, increases, the first algorithm does better, and as k is small, the latter is better.

LP Rounding

- The next steps:
- We have two algorithms. One guarantees an approximation ratio of at least 1/2.
- The other guarantees an approximation ratio of at least
  $1 - (1 - 1/k)^k$.
- We can use both to achieve an approximation ratio of at least 3/4.
- Idea 1 : Run both algorithms and pick the best solution.
- Idea 2 : Toss a fair coin and pick an algorithm based on the outcome.....
- In both cases, we can show that the expected performance ratio will be at least 3/4.

LP Rounding

- We have two algorithms. One guarantees an approximation ratio of at least 1/2. The other guarantees an approximation ratio of at least $1 - (1 - 1/k)^k$.
- We can use both to achieve an approximation ratio of at least 3/4.
- Idea 2 : Toss a fair coin and pick an algorithm based on the outcome.....
- In this case, the expected number of clauses satisfied is $(n_1+n_2)/2$.
  - Here, n_1 and n_2 refer to the expected number of clauses satisfied by the first and the second algorithms.

LP Rounding

- We have two algorithms. One guarantees an approximation ratio of at least 1/2. The other guarantees an approximation ratio of at least $1 - (1 - 1/k)^k$.
- Idea 2 : Toss a fair coin and pick an algorithm based on the outcome.....
- In this case, the expected number of clauses satisfied is $(n_1+n_2)/2$.
  - Here, n_1 and n_2 refer to the expected number of clauses satisfied by the first and the second algorithms.
- We will show that $(n_1+n_2)/2 >= ¾ \Sigma_i v_i$

LP Rounding

- In this case, the expected number of clauses satisfied is $(n_1+n_2)/2$.
  - Here, n_1 and n_2 refer to the expected number of clauses satisfied by the first and the second algorithms.
- We will show that $(n_1+n_2)/2 >= ¾ \Sigma_j v_j$
- Focus on clauses that have k literals. Then,
- $n_1 = \Sigma_k \Sigma_{C_j \text{ has k literals}} (1 – 2^{-k}) >= \Sigma_k \Sigma_{C_j \text{ has k literals}} (1 – 2^{-k}). v_j.$

- We know that $n2 >= \Sigma_k \Sigma_{C_j \text{ has k literals}} \beta_k. v_j$ where $\beta_k = 1 - (1 – 1/k)^k$

LP Rounding

- We will show that $(n_1+n_2)/2 >= ¾ \Sigma_j v_j$
- Focus on clauses that have k literals. Then,
- $n_1 = \Sigma_k \Sigma_{C_j \text{ has k literals}} (1 – 2^{-k}) >= \Sigma_k \Sigma_{C_j \text{ has k literals}} (1 – 2^{-k}). v_j.$

- We know that $n_2 >= \Sigma_k \Sigma_{C_j \text{ has k literals}} \beta_k. v_j$ where $\beta_k = 1 - (1 – 1/k)^k$
- So, $(n_1+n_2)/2 >= \Sigma_k \Sigma_{C_j \text{ has k literals}} ½ [(1 – 2^{-k}) + \beta_k] . v_j.$
- We prove that $(1 – 2^{-k}) + \beta_k$ is at least 3/2.
- Ending the proof of the required claim.

Approximate Counting

- We now focus on a new topic called approximate counting.
- The idea here is to see how many objects satisfy a given condition.
- Examples:
  - Count the number of spanning trees of a given graph.
  - Count the number of matchings of a given graph.
  - Count the number of truth assignments that satisfy a formula in DNF.
- As can be seen, the first example has a polynomial time algorithm while the other two do not.

Approximate Counting

- We start with studying how to count the number of truth assignments that satisfy a formula in DNF.
- A formula in DNF is of the form C_1 V C_2 V ... V C_m where each clause C_i is of the form ($L_1 \wedge L_2 \wedge ... \wedge L_j$).
- The L's are called literals and each literal is either a variable X_k or its negation X_k'.
- The problem has applications in network design and realibility.

Approximate Counting

- We start with studying how to count the number of truth assignments that satisfy a formula in DNF.
- A formula F in DNF is of the form $C_1 \vee C_2 \vee \ldots \vee C_m$ where each clause C_i is of the form $(L_1 \wedge L_2 \wedge \ldots \wedge L_{r_i})$.
- The L's are called literals and each literal is either a variable X_k or its negation X_k'.
- A truth assignment is said to satisfy F if some clause in F is true under the assignment.
- We are interested in #F – the number of distinct satisfying assignments of F.
- While no deterministic algorithms are known, we focus on randomized algorithms to get as close to the answer as possible.

Approximate Counting

- We abstract the problem as follows.
- Let U be a finite set and f : U -> {0, 1} be a Boolean function.
- Define G = { u | f(u) = 1}.
- We assume that given a u in U, f(u) is easy to compute.
- We also assume that it is possible to sample uniformly at random from U.
- We want to estimate the size of G.

Approximate Counting

- The method we study is called the Monte Carlo method.
- Choose N independent samples uar from U, say u_1, u_2, ..., u_N.
- Evaluate f at each of these samples and count the number of instances that f evaluates to 1.
- Use this count to estimate G.
- Details follow.

Approximate Counting

- The method we study is called the Monte Carlo method.
- Choose N independent samples uar from U, say u_1, u_2, ..., u_N.
- Evaluate f at each of these samples and count the number of instances that f evaluates to 1.
- Use this count to estimate G.
- Let Y_i be a random variable that indicates whether u_i in G.
- Define another random variable $Z = |U|. \sum_i Y_i/N$.
- Verify that $E[Z] = |G|$.

Approximate Counting

- The method we study is called the Monte Carlo method.
- Choose N independent samples uar from U, say u_1, u_2, ..., u_N.
- Evaluate f at each of these samples and count the number of instances that f evaluates to 1.
- Use this count to estimate G.
- Let Yi be a random variable that indicates whether ui in G.
- Define another random variable $Z = |U|. \sum_i Y_i/N$.
- Verify that $E[Z] = |G|$.
- $E[Z] = (|U|/N). \sum_i E[Y_i] = (|U|/N). \sum_i |G|/|U| = |G|$.

Approximate Counting

- The method we study is called the Monte Carlo method.
- Choose N independent samples uar from U, say u_1, u_2, ..., u_N.
- Evaluate f at each of these samples and count the number of instances that f evaluates to 1.
- Use this count to estimate G.
- Let Yi be a random variable that indicates whether ui in G.
- Define another random variable $Z = |U|. \sum_i Y_i/N$.
- Verify that $E[Z] = |G|$.
- $E[Z] = (|U|/N). \sum_i E[Y_i] = (|U|/N). \sum_i |G|/|U| = |G|$.
- We hope that since Z has an expectation that is usfeul for us, Z can be used as an estimator for |G|.

Approximate Counting

- Define another random variable $Z = |U|. \sum_i Y_i/N$.
- Verify that $E[Z] = |G|$.
- $E[Z] = (|U|/N). \sum_i E[Y_i] = (|U|/N). \sum_i |G|/|U| = |G|$.
- We hope that since Z has an expectation that is usfeul for us, Z can be used as an estimator for |G|.
- In other words, we hope that Z has a value close to |G|.
- Indeed, an application of Chernoff bounds again.

Approximate Counting

- Let r = |G|/|U|. Then, with probability at least $1 - \delta$, the Monte Carlo estimation produces a result that is within ε of the actual value.
- Recall that each Y_i is a Bernoulli random variable and the Y_i's are iid.
- Define $Y = \sum_i Y_i$ and realize that $Z = |U|Y/N$.
- $E[Y] = Nr$.
- Using Chernoff bounds, $Pr((1-\varepsilon)|G| <= Z <= (1+\varepsilon)|G|)$
- $= Pr((1-\varepsilon)|G| <= |U|Y/N <= (1+\varepsilon)|G|)$

  $= Pr((1-\varepsilon)Nr <= Y <= (1+\varepsilon) Nr)$

  $>= 1 - 2exp\{-\varepsilon^2 Nr/4)$.

Approximate Counting

- Let r = |G|/|U|. Then, with probability at least $1 - \delta$, the Monte Carlo estimation produces a result that is within ε of the actual value with $N >= 4 \ln(2/\delta)/\varepsilon^2 r$.
- Define $Y = \sum_i Y_i$ and realize that $Z = |U|Y/N$. $E[Y] = Nr$.
- Using Chernoff bounds, $Pr((1-\varepsilon)|G| <= Z <= (1+\varepsilon)|G|)$
- $= Pr((1-\varepsilon)|G| <= |U|Y/N <= (1+\varepsilon)|G|)$

  $= Pr((1-\varepsilon)Nr <= Y <= (1+\varepsilon) Nr)$

  $>= 1 - 2\exp\{- \varepsilon^2 Nr/4\}$.
- With the choice of N, $\exp\{- \varepsilon^2 Nr/4\} = \exp\{-\ln (2/\delta)\})$.
- So, the required probability is at least $1 - \delta$.

Approximate Counting

- How good is our scheme?
- Notice that the scheme requires N samples to be drawn.
- But $N = O(1/r)$ where $r = |G|/|U|$.
- If $|G|$ is small, then N is large.
- N is not only large, but is as large as 1/r implying that N can get close to being exponentially large.

Approximate Counting

- How good is our scheme?
- Notice that the scheme requires N samples to be drawn.
- But N = O(1/r) where r = |G|/|U|.
- If |G| is small, then N is large.
- N is not only large, but is as large as 1/r implying that N can get close to being exponentially large.
- The trouble is not with the Chernoff bound but with uar sampling.
- For cases with r too small, fetching useful samples is difficulty. So, needs more trails.

Importance Sampling

- Do not rely on uniform sampling but use skewed sampling.
- Skew towards useful samples.
- Details follow.

Importance Sampling

- One aspect of importance sampling is to increase the value of r = |G|/|U|.
- With a large r, we will avoid the danger of N = O(1/r) getting large when r is small.
- The approach we study is quite generic.
- Let V be a finite universe.
- We are given m subsets H_1, H_2, ..., H_m of V such that

1) For all i, $|H_i|$ can be obtained in polynomial time.

2) It is possible to sample uniformly at random from any H_i.

3) For all v in V, it is possible to determine if v in H_i in polynomial time.

Importance Sampling

- One aspect of importance sampling is to increase the value of r = |G|/|U|.
- With a large r, we will avoid the danger of N = O(1/r) getting large when r is small.
- The approach we study is quite generic.
- Let V be a finite universe.
- We are given m subsets H_1, H_2, ..., H_m of V such that

1) For all i, $|H_i|$ can be obtained in polynomial time.

2) It is possible to sample uniformly at random from any H_i.

3) For all v in V, it is possible to determine if v in H_i in polynomial time.

- The goal is to estimate the size of $H = U_i H_i$.

Importance Sampling

- Let V be a finite universe.
- We are given m subsets H_1, H_2, ..., H_m of V such that

1) For all i, $|H_i|$ can be obtained in polynomial time.

2) It is possible to sample uniformly at random from any H_i.

3) For all v in V, it is possible to determine if v in H_i in polynomial time.

- The goal is to estimate the size of $H = U_i H_i$.
- One possibility is to use the inclusion-exclusion principle.
- But there are too many terms in the formula.

Importance Sampling

- Let V be a finite universe. We are given m subsets H_1, H_2, ..., H_m of V such that:

1) For all i, $|H_i|$ can be obtained in polynomial time.

2) It is possible to sample uar from any H_i.

3) For all v in V, it is possible to determine if v in H_i in polynomial time.

- The goal is to estimate the size of $H = U_i H_i$.
- How is the DNF Counting problem similar?
- The universe V is the all possible assignments.
- Each H_i corresponds to the assignments that satisfy C_i
- What is $|H_i|$ for a given i?

Importance Sampling

- Let V be a finite universe. We are given m subsets H_1, H_2, ..., H_m of V such that:

  1) For all i, $|H_i|$ can be obtained in polynomial time.

  2) It is possible to sample uar from any H_i.

  3) For all v in V, it is possible to determine if v in H_i in polynomial time.

- The goal is to estimate the size of $H = U_i H_i$.
- How is the DNF Counting problem similar?
- The universe V is the all possible assignments.
- Each H_i corresponds to the assignments that satisfy C_i
- What is $|H_i|$ for a given i? Exactly 2^{n-r_i}.
- Easy to sample too. Fix the truth values of r_i variables, others can be set uar.

Importance Sampling

- Let V be a finite universe. We are given m subsets H_1, H_2, ..., H_m of V such that:

  1) For all i, $|H_i|$ can be obtained in polynomial time.

  2) It is possible to sample uar from any H_i.

  3) For all v in V, it is possible to determine if v in H_i in polynomial time.

- The goal is to estimate the size of $H = U_i H_i$.
- How is the DNF Counting problem similar?
- The universe V is the all possible assignments.
- Each H_i corresponds to the assignments that satisfy C_i
- What is $|H_i|$ for a given i? Exactly 2^{n-r_i}.
- Easy to sample too. Fix the truth values of ri variables, others can be set uar.
- Can also check if a given v in V is in H_i.

Importance Sampling

- Let V be a finite universe. We are given m subsets H_1, H_2, ..., H_m of V such that:

  1) For all i, $|H_i|$ can be obtained in polynomial time.

  2) It is possible to sample uar from any H_i.

  3) For all v in V, it is possible to determine if v in H_i in polynomial time.

- The goal is to estimate the size of $H = U_i H_i$.
- How is the DNF Counting problem similar?
- The universe V is the all possible assignments.
- Each H_i corresponds to the assignments that satisfy C_i
- What is $|H_i|$ for a given i? Exactly 2^{n-r_i}.
- Easy to sample too. Fix the truth values of r_i variables, others can be set uar.
- Can also check if a given v in V is in H_i.
- And, the DNF counting problem is to estimate |H|.

Importance Sampling

- Define a multiset U as the union of H_i's where the multiset keeps multiple copies of each element.
- Viewed differently, U can be seen as consisting of tuples (v, i) such that v is in H_i.
- Hence, U = { (v, i) | v is in H_i}.
- Now, $|U| = \Sigma_i |H_i| >= |H|$.
- We also define for every v in V, the coverage set of v as
- cov(v) = { (v,i) | (v,i) in U}
- Notice that for a given v, the size of cov(v) is the number of H_i's that contain v_i.

Importance Sampling

- Viewed differently, U can be seen as consisting of tuples (v, i) such that v is in H_i.
- Hence, U = { (v, i) | v is in H_i}. Now, |U| = $\sum_i |H_i|$ >= |H|.
- We also define for every v in V, the coverage set of v as
- cov(v) = { (v,i) | (v,i) in U}. Notice that for a given v, the size of cov(v) is the number of H_i's that contain v_i.
- The following observations hold wrt cov()

  1) The number of coverage sets is exactly |H|.

  2) The coverage sets partition U, i.e. U = $U_{v \text{ in } H}$ cov(v).

  3) As a result of 2, |U| = $\sum_{v \text{ in } H} |cov(v)|$.

  4) For all v in H, |cov(v)| is at most m.

Importance Sampling

- Based on the above definitions, for each coverage set, we define a canonical element as follows.
- Define a function f: U -> {0,1} as f((v,i)) = 1 if i = min{j | v in H_j} and 0 otherwise.
- Define G = { (v,j) in U | f((v,j) = 1}.
- Notice that |G| = |H|.

Importance Sampling

- Based on the above definitions, for each coverage set, we define a canonical element as follows.
- Define a function f: U -> {0,1} as f((v,i)) = 1 if i = min{j | v in H_j} and 0 otherwise.
- Define G = { (v,j) in U | f((v,i) = 1}.
- Notice that |G| = |H|.
- Further, r = |G|/|U| >= 1/m now. As,
  - $|U| = \sum_{v \text{ in } H} |cov(v)| <= \sum_{v \text{ in } H} m <= m\,|H| = m\,|G|$.

- Now, use the Monte-Carlo technique of sampling on U.
- Same calculations as earlier hold now. Except that the time taken is O(1/r) = O(polynomial).
- Only have to show how to sample uar from U.

Importance Sampling

- Sampling uar from U is done in two steps as follows:

  1)Choose an i between 1 and m such that Pr[i is chosen] = $|H_i|/|U|$.

  2)Once an i is chosen, choose a v in H_i u.a.r.

- Can claim that the pair (v,i) is now uar over U.

Importance Sampling

- Why did this sampling procedure work?
- Cannot sample uar from V and quickly get a good approximation.
- However, using U in the importance sampling experiment is equivalent to sampling a random satisfying truth assignment from a randomly chosen clause.
- The probability we choose a particular assignment now depends on the number of clauses this assignment satisfies.
- However, since G is defined on the basis of canonical tuples, only one choice of truth assignment and the correct clause makes it count correctly towards the size of G.

Some Definitions

- With respect to approximate counting algorithms, one often uses the following notion.
- A polynomial randomized approximation scheme (PRAS) for a counting problem Π is a randomized algorithm A running in polynomial time (in n) such that on any instance I of P and a real number $\varepsilon > 0$, produces an output A(I) such that $\Pr[(1-\varepsilon)\#I \leq A(I) \leq (1+\varepsilon)\#I] \geq 3/4$.

- A PRAS scheme for a problem Π is called a fully polynomial randomized approximation scheme (FPRAS) if in addition the runtime of the algorithm A is bounded by a polynomial in n and $1/\varepsilon$.

Some Definitions

- A polynomial randomized approximation scheme (PRAS) for a counting problem Π is a randomized algorithm A running in polynomial time (in n) such that on any instance I of P and a real number $\varepsilon > 0$, produces an output A(I) such that $\Pr[(1-\varepsilon)\#I \leq A(I) \leq (1+\varepsilon)\#I] \geq 3/4$.

- A PRAS scheme for a problem Π is called a fully polynomial randomized approximation scheme (FPRAS) if in addition the runtime of the algorithm A is bounded by a polynomial in n and $1/\varepsilon$.
- If the probability above can be made arbitrarily small to a given δ, while the runtime is poly(n, $1/\varepsilon$, δ), then such a scheme is called as an (ε,δ)-FPRAS.

Some Definitions

- A problem is in the class #P if the corresponding decision problem is in NP.
- More formally, a problem is in #P if there is a non-deterministic polynomial time Turing machine that for any instance I has a number of accepting computations that is exactly equal to the number of distinct solution to instance I.
- Further, a problem Π is said to be #P-complete if for any problem Π' in #P, Π' can be reduced to Π in polynomial time.