# Randomized Algorithms

C.R. Subramanian

The Institute of Mathematical Sciences, Chennai.

Expository talk presented at the Research Promotion Workshop on "Introduction to Geometric and Graph Algorithms" at PSG College of Technology, Coimbatore, January 6-8, 2011.

## Deterministic Algorithms : Characteristics

- $\pi$ - a computational problem ; Eg : Sorting.

- $A$ - a deterministic algorithm to solve $\pi$. Eg : Selection Sort.

- At every point during an execution of algorithm $A$ over $I$, the next move of $A$ is uniquely well-defined.

- The execution and running time, intermediate steps and the final output computed are the same during each execution of $A$ over $I$.

- The course followed by the algorithm does not vary with execution as long as the input $I$ remains the same.

# Randomized Algorithms : Characteristics

- $RA$ - a randomized algorithm to solve $\pi$.

- At every point during an execution of algorithm $RA$ over $I$, the next move of $A$ can possibly be determined by employing randomly chosen bits and is not uniquely well-defined.

- The execution and running time, intermediate steps and the final output computed could possibly vary for different executions of $RA$ over the same $I$.

- The course followed by the algorithm varies for different executions even if the input $I$ remains the same.

# Why Randomization ?

- Randomness often helps in significantly reducing the work involved in determining a correct choice when there are several but finding one is very time consuming.

- Reduction of work (and time) can be significant on the average or in the worst case.

- Randomness often leads to very simple and elegant approaches to solve a problem or it can improve the performance of the same algorithm.

- Risk : loss of confidence in the correctness. This loss can be made very small by repeated employment of randomness.

- Assumes the availability of truly unbiased random bits which are very expensive to generate in practice.

# Some Tail Inequalities

- $X$ - random variable : $\mu = E[X]$; $Var(X) = E[X^2] - \mu^2$.
- Markov : $X$ is nonnegative ;
- For every $t > 0$, $\mathbf{Pr}(X \geq t) \leq \mu/t$.
- Chebyschev : $\forall t > 0$, $\mathbf{Pr}(|X - \mu| \geq t) \leq Var(X)/t^2$.
- $\forall \epsilon \in (0, 1]$, $\mathbf{Pr}(|X - \mu| \geq \epsilon\mu) \leq Var(X)/\epsilon^2\mu^2$.
- Chernoff : $X = X_1 + \ldots + X_n$; $X_i \in \{0, 1\}$. independent.
- $\forall \epsilon \in (0, 1]$, $\mathbf{Pr}(X \leq \mu(1 - \epsilon)) \leq e^{-\epsilon^2\mu/2}$.
- $\forall \epsilon \in (0, 1]$, $\mathbf{Pr}(X \geq \mu(1 + \epsilon)) \leq e^{-\epsilon^2\mu/3}$.

## QuickSort($A, p, q$):

- If $p \geq q$, EXIT.

- $s \leftarrow$ correct position of $A[p]$ in the sorted order.

- Move the "pivot" $A[p]$ into position $s$.

- Move the remaining elements into "appropriate" positions.

- Quicksort($A, p, s - 1$);

- Quicksort($A, s + 1, q$).

# Worse-case Complexity of QuickSort :

- $T(n)$ = Worst-case Complexity of QuickSort on an input of size $n$. Only comparisons are counted.

- $T(n) = \max\{T(\pi) : \pi \text{ is a permutation of } [n]\}$.

- $T(n) = \Theta(n^2)$.

- Worst case input is : $\pi = \langle n, n - 1, \ldots, 1 \rangle$.

- There exist inputs requiring $\Theta(n^2)$ time.

## Randomized Version : RandQSort($A, p, q$):

- If $p \geq q$, EXIT.

- Choose uniformly at random $r \in \{p, \ldots, q\}$.

- $s \leftarrow$ correct position of $A[r]$ in the sorted order.

- Move randomly chosen pivot $A[r]$ into position $s$.

- Move the remaining elements into "appropriate" positions.

- RandQSort($A, p, s-1$);

- RandQSort($A, s+1, q$).

## Analysis of RandQSort :

- Every comparison is between a pivot and another element.

- two elements are compared at most once.

- rank of an element is the position in the sorted order.

- $x_i$ is the element of rank $i$. $S_{i,j} = \{x_i, \ldots, x_j\}$.

- $X_{i,j} = 1$ if $x_i$ and $x_j$ are ever compared and 0 otherwise.

- $E[T(\pi)] = E[\sum_{i<j} X_{i,j}] = \sum_{i<j} E[X_{i,j}]$.

- $E[X_{i,j}] = \frac{2}{j-i+1}$.

- $E[T(\pi)] = \sum_{i<j} \frac{2}{j-i+1} \leq 2nH_n = \Theta(n(\log n))$.

# Example of randomness improving the efficiency :

- Analysis holds for every permutation $\pi$.
- $T(n)$ = Maximum value of the Expected Time Complexity of RandQSort on an input of size $n$.
- $T(n) = \max\{E[T(\pi)] : \pi$ is a permutation of $[n]\}$.
- $T(n) = \Theta(n(\log n))$.
- For every $\pi$, $\mathbf{Pr}(T(\pi) > 8nH_n) \leq 1/4$.
- introducing randomness very likely improves the efficiency.
- An example of a Las Vegas algorithm : always correct but running time varies but with possibly poly *expected* time.

## Verifying matrix multiplication :

- $A, B, C \in F^{n \times n}$;    Goal : To verify if $AB = C$.

- direct approach - $O(n^3)$ time.

- algebraic approach - $O(n^{2.376})$ time.

- Randomized Alg :

-      Choose *u.a.r.* $r \in \{0, 1\}^n$ and check if $ABr = Cr$.

-      If so, output YES, otherwise output NO.

- If $AB \neq C$, then $\mathbf{Pr}(ABr = Cr) \leq 1/2$.

- requires $O(n^2)$ time.

- An example of a Monte-Carlo algorithm : can be incorrect but guaranteed running time and with a guarantee of confidence.

## Las Vegas vs Monte-Carlo :

- Las Vegas $\rightarrow$ Monte-Carlo

- $A$ - Las Vegas algo with $E[T_A(I)] \leq poly(n)$ for every $I$.

- By incorporating a counter which counts every elementary step into $A$ and stopping after, say, $4poly(n)$ steps, one gets a poly time Monte-Carlo algorithm $B$ with a guaranteed confidence of at least $3/4$.

- Monte-Carlo $\rightarrow$ Las Vegas

- $A$ - Monte-Carlo alg with $poly(n)$ time and $1/poly(n)$ success probability. Suppose correctness of output can be verified in $poly(n)$ time.

- By running the alg $A$ repeatedly (with independent coin tosses) until one gets a correct solution, we get a Las Vegas algo with poly expected time.

## Randomization provably helps

- A simple counting problem (by CRS) :

- $A[1 \ldots n]$-array with $A_i \in \{1, 2\}$ for every $i$.

- $f(x) = \text{freq}(x) \geq n/5$ for each $x \in \{1, 2\}$.

- Goal : Given $x \in \{1, 2\}$ and an $\epsilon > 0$,

- determine $ans$ : $ans \in [(1 - \epsilon)f(x), (1 + \epsilon)f(x)]$.

- Any deter. alg needs $\Omega(n)$ queries in the worst case for $\epsilon = 1/10$.

- $\exists$ rand. alg with $O(\log n)$ queries for every fixed $\epsilon$.

# Randomization provably helps

- **RandAlg($A, x, \epsilon$) :**

- $m = 20(\log n)/\epsilon^2$ ; $c = 0$.

- **for** $i = 1, \ldots, m$ **do**

- Choose uniformly at random $j \in \{1, \ldots, n\}$.

- **if** $A[j] = x$ **then** increment $c$.

- **endfor**

- Return $ans = nc/m$.

- **end**

## Randomization provably helps

- **Analysis of RandAlg($A, x, \epsilon$) :**

- $X_i = 1$ if $A[j] = x$ for $j$ chosen in the $i$th-iteration.

- $c = \sum_i X_i; \qquad E[X_i] = f(x)/n.$

- $\mu = E[c] = mf(x)/n \geq m/5. \qquad E[ans] = f(x).$

- $\mathbf{Pr}(c \notin [(1 - \epsilon)\mu, (1 + \epsilon)\mu]) \leq 2e^{-\epsilon^2\mu/3} = o(n^{-1}).$

- $(1 - \epsilon)f(x) \leq ans \leq (1 + \epsilon)f(x)$ with probability $1 - o(1).$

- No. of queries $= O((\log n)/\epsilon^2).$

- No. of queries $= O(1)$ with success probability $\geq 3/4.$

# Unbiased Estimator

- $A$ is a randomized algorithm to approximate $\#I$.

- $A$ outputs $X$ such that $\mu = E[X] = \#I$.

- $\mathbf{Pr}(X \notin [(1 \pm \epsilon)\mu]) \leq Var(X)/\epsilon^2\mu^2$ by Chebyshev.

- $Var(X) = O(\mu) \Rightarrow reqd.prob = O(\epsilon^{-2}\mu^{-1})$.

- Example above : $Var(X) \leq \mu$, helps us !

- Often, $X$ is not so nicely defined and $Var(X)$ may not be small compared to $\mu^2$.

# Boosting Success Probability - I

- Run $m$ independent trials of $A(I, \epsilon)$.

- Take $ans$ to be the numerical average of $\{X_1, \ldots, X_m\}$.

- $E[ans] = \mu$ and $Var(ans) = Var(X)/m$.

- $\mathbf{Pr}(ans \notin [(1 \pm \epsilon)\mu]) \leq Var(X)/m\epsilon^2\mu^2$.

- $\mathbf{Pr}(\text{success}) \geq 3/4$ provided $m \geq 4E[X^2]/\epsilon^2\mu^2$.

- a good approximation efficiently computable.

# Boosting success probability - II

- $A$ is a randomized algorithm to approximate $\#I$.

- $A$ runs in time $poly(n, 1/\epsilon)$ and outputs $ans$ :

- $\mathbf{Pr}((1 - \epsilon)(\#I) \leq ans \leq (1 + \epsilon)(\#I)) \geq 1/2 + \delta$.

- Run $m$ independent trials of $A(I, \epsilon)$.

- Take $ans$ to be the median of $\{ans_1, \ldots, ans_m\}$.

- $\mathbf{Pr}\left((1 - \epsilon)(\#I) \leq ans \leq (1 + \epsilon)(\#I)\right) \geq 1 - e^{-\delta^2 m/2}$.

- $\mathbf{Pr}(\text{success}) = 1 - o(n^{-1})$ provided $m \geq 4(\log n)/(\delta^2)$.

- a good approximation efficiently computable.

# Approximating Frequency Moments

- Given $A = (a_1, \ldots, a_m)$, $a_j \in \{1, \ldots, n\}$.

- $m_i$ = frequency of $i$ in $A$, $i \in \{1, \ldots, n\}$.

- Determine $F_k = \sum_i m_i^k$ using "small" space.

- $F_0$ = number of distinct elements in $A$.

- $F_1$ = length of the sequence $A$ ; $F_2$ = repeat rate of $A$.

- Determining $F_k$ arises in Data Mining.

- Suppose we want to collect some statistical information from

- a large stream of data without having to store the data.

- $\Omega(n)$ bits needed for any deter. alg approximating $F_k$ within a ratio of $1 \pm 0.1$.

# Approximating $F_2$ - algorithm (Alon, Matias, Szegedy)

- $V = \{v_1, \ldots, v_h\}$, $h = O(n^2)$, each $v_i$ - a $n$-vector of $\pm 1$.

- $V$ is four-wise independent. For $v \in_R V$, $\forall i_1 \leq \ldots \leq i_4$, $\forall (\epsilon_1, \ldots \epsilon_4) \in \{-1, 1\}^4$, $\mathbf{Pr}(\forall j, v(i_j) = \epsilon_j) = 1/16$.

- Choose $p \in_R \{1, \ldots, h\}$ and store it using $O(\log n)$ bits.

- $v_p(i)$ can be found (for a given $i$) using only $O(\log n)$ bits.

- $Z = \sum_i \epsilon_i m_i$. $Z$ can be computed in one pass using $O(\log n + \log m)$ bits.

- Compute $X = Z^2$. space $= O(\log m + \log n)$.

- Take $s_1 = 16/\lambda^2$ independent samples $X_j = X$ and take their average $Y$.

- Take $s_2 = 2\log(1/\epsilon)$ independent samples $Y_i = Y$ and output their median.

# Approximating $F_2$ - analysis

- $E[X] = E[(\sum_i \epsilon_i m_i)^2] = \sum_i m_i^2 = F_2$.

- $E[X^2] = \sum_i m_i^4 + 6 \sum_{i<j} m_i^2 m_j^2$.

- $Var(X) = E[X^2] - E[X]^2 = 4 \sum_{i<j} m_i^2 m_j^2 \leq 2F_2^2$.

- $\mathbf{Pr}(|Y_i - F_2| \geq \lambda F_2) \leq \frac{2F_2^2}{s_1 \lambda^2 F_2^2} \leq 1/8$.

- $\mathbf{Pr}(|Y - F_2| \geq \lambda F_2) \leq \epsilon$.

- Total space complexity $= O(\log(1/\epsilon)(\log n + \log m)/\lambda^2)$ bits.

# Some examples of counting problems

- Given a connected $G = (V, E)$, determine the number of spanning tress of $G$.

- Given $G$, determine the number of perfect matchings in $G$.

- Given $G$, determine the number of 3-colorings of $G$.

- Given a boolean formula $F$ on $n$ boolean variables, determine the number of satisfying assignments of $F$.

# Class $\#P$

- $M$ is a NP machine.

- $f_M : \Sigma^* \to \mathcal{N}$ such that

- $f_M(I) =$ number of accepting paths of $M$ on $I$, $\quad \forall \, I \in \Sigma^*$.

- Counting version denoted by $\#M$ :

- $\qquad$ Given $I \in \Sigma^*$, determine $f_M(I)$.

- $\#P = \{\#M : M \text{ is a NP machine }\}$.

# #*P*-complete problems

- $\pi_1, \pi_2 \in \#P$.

- $\pi_1 \; \alpha_T \; \pi_2$  if there is a deterministic poly time algorithm for solving $\pi_1$ using a polynomial number of queries to an algorithm for solving $\pi_2$.

- $\pi$ is #*P*-hard if

- $\pi_1 \; \alpha_T \; \pi$  for every $\pi_1 \in \#P$.

- $\pi$ is #*P*-complete if

- $\pi \in \#P$ and $\pi$ is #*P*-hard.

- if a #*P*-complete $\pi$ can be solved in poly time,

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

## Some examples of easy counting problems

- COUNT the number of 2-colorings of a given $G$.

- $O(n + m)$ time ; Compute connected components.

- COUNT the number of spanning trees of a connected $G$.

- reduces to computing the determinant of a suitable matrix.

- arborescence - directed rooted tree with a unique path ;

- COUNT the number of arborescences of a directed $G$.

- COUNT the number of Eulerian circuits of a directed $G$.

- $\#PM$ for planar graphs $G$;

- reduces to computing the determinant of a suitable matrix.

# Some examples of hard counting problems

- **1.** DNF satisfying assignments
- Given $C = C_1 \vee C_2 \vee \ldots \vee C_m$,
- each $C_j$ - conjunction of literals of $n$ variables.
- Count the number of satisfying assignments.
- decision version trivially solvable.

- **2.** #Perfect Matchings
- Given arbitrary $G$,
- Count the number of perfect matchings.
- decision version polynomial time solvable.

# Some examples of hard counting problems

- **3.** #-3-colorings
- Given arbitrary $G$, Count the number of 3-colorings.
- decision version NP-complete. Counting is hard
- for every fixed $k \geq 3$ or if $G$ is bipartite.
- **4.** Permanent computation
- $A$ - $n \times n$, $(0, 1)$-matrix.
- $Perm(A) = \sum_{\sigma \in S_n} \Pi_i A_{i, \sigma(i)}$.
- same as #PMs of $G = (U \cup V, E)$, $|U| = |V| = n$.

# More examples of hard counting problems

- **5.** #hamilton cycles of $G$.

- **6.** #acyclic orientations of $G$.

- **7.** #cycles in a directed graph.

- **8.** #cliques in a graph.

- **9.** #maximal cliques in a graph.

- **10.** #independent sets in a graph.

# FPRAS

- Fully Polynomial Randomized Approximation Scheme

- Algorithm $A(x, \epsilon)$ and produces *ans* :

- $\mathbf{Pr}(ans \in (1 \pm \epsilon)f(x)) \geq 1/2 + \delta$.

- time bound is $poly(|x|, 1/\epsilon)$.

- $(1 \pm \epsilon)f(x) =_{def} [(1 - \epsilon)f(x), (1 + \epsilon)f(x)]$.

- ($\pi$ is NPC and $NP \nsubseteq RP$) $\implies$ ($\#\pi$ admits no FPRAS.)

# Counting DNF assignments

- $C = C_1 \vee C_2 \vee \ldots \vee C_m$. $n$ boolean variables.

- $\#C_j =$ no. of satisfying assignments.

- Eg : $C_j = x_1 \wedge \overline{x_3} \wedge x_8$. $\#C_j = 2^{n-3}$.

- Goal : determine $\#C$.

- Obs 1: For every $S \subset \{1, \ldots, m\}$, no. of assignments satisfying $C_j$ for each $j \in S$ can be computed in poly time.

- for every $j$, let $N_j = \sum_{|S|=j} \#(\wedge_{i \in S} C_i)$.

- PIE : $\#C = N_1 - N_2 + \ldots + (-1)^{m+1} N_m$.

- takes exponential (in $m$) time.

# Randomized DNF Counting-1

- **RandAlg-1($C, \epsilon$) :**

- $M = (n + m)^{O(1)}$ ; $c = 0$ ; $V = \{x_1, \ldots, x_n\}$.

- **for** $i = 1, \ldots, M$ **do**

- Choose uniformly at random $f : V \to \{T, F\}$.

- **if** $f$ satisfies $C$ **then** increment $c$.

- **endfor**

- Return $ans = 2^n c / M$.

- **end**

# Randomized DNF Counting-1

- **Analysis of RandAlg-1($C, \epsilon$) :**

- $X_j = 1$ if $f$ satisfies $C$ for $f$ chosen in the $j$th-iteration.

- $c = \sum_i X_i$;     $E[X_i] = (\#C)/2^n$.

- $\mu = E[c] = M(\#C)/2^n$.     $E[ans] = \#C$.

- $\mathbf{Pr}(c \notin [(1-\epsilon)\mu, (1+\epsilon)\mu]) \leq 2e^{-\epsilon^2\mu/3}$.

- $\mathbf{Pr}\left((1-\epsilon)\#C \leq ans \leq (1+\epsilon)\#C\right) \geq 1 - 2e^{-\epsilon^2\mu/3}$.

- $\mathbf{Pr}(\text{success}) \geq 1/2$ provided $M \geq 6 \cdot 2^n/(\epsilon^2 \#C)$.

- poly time possible only if $\#C \geq 2^n/n^{O(1)}$.

# Randomized DNF Counting-1

- **Analysis of RandAlg-1($C, \epsilon$) :**

- Major problem is $\#C/2^n$ can be very small.

- Essentially, it is the following problem.

- A coin with unknown bias $p$ is given. One is allowed to repeatedly and independently toss the coin.

- How many trials needed to "estimate" $p$ ? Expected no. of trials to get a HEAD is $1/p$.

- We improve $p$ by reducing the size of the search space.

# Randomized DNF Counting-2

- **RandAlg-2($C, \epsilon$) :** (Karp and Luby)

- **for** $j = 1, \ldots, m$ compute $N_j = \#C_j$.

- $N = \sum_j N_j$ ; $M = (n+m)^{O(1)}$ ; $c = 0$ ;

- **for** $i = 1, \ldots, M$ **do**

- Choose a random $r \in \{1, \ldots, m\}$
  with $\mathbf{Pr}(r = j) = N_j/N$ for each $j$.

- Choose uniformly at random a $f : V \rightarrow \{T, F\}$
  which satisfies $C_r$.

- **if** $f$ satisfies $C_r$ but does not satisfy any $C_j$
  for $j < r$, **then** increment $c$.

- **endfor**

- Return $ans = Nc/M$.

- **end**

# Randomized DNF Counting-2

- **Analysis of RandAlg-2($C, \epsilon$) :**

- $X_i = 1$ if the $i$th-iteration is a success ($c$ is incremented).

- $c = \sum_i X_i; \qquad E[X_i] = (\#C)/N.$

- $\mu = E[c] = M(\#C)/N \geq M/m. \qquad E[ans] = \#C.$

- $\mathbf{Pr}(c \notin [(1-\epsilon)\mu, (1+\epsilon)\mu]) \leq 2e^{-\epsilon^2\mu/3}.$

- $\mathbf{Pr}\left((1-\epsilon)\#C \leq ans \leq (1+\epsilon)\#C\right) \geq 1 - 2e^{-\epsilon^2\mu/3}.$

- $\mathbf{Pr}(\text{success}) = 1 - o(n^{-1})$ provided $M \geq 4m(\log n)/(\epsilon^2).$

- a good approximation computable in poly time.

# To count, just sample

- To estimate $|\Omega|$, embed $\Omega$ into a bigger $\mathcal{U}$ :

- ($i$) $|\mathcal{U}|$ can be determined exactly and efficiently.

- ($ii$) efficient testing of membership in $\Omega$ is possible.

- ($iii$) a uniformly random sample from $\mathcal{U}$ can be generated efficiently.

- ($iv$) the ratio $\frac{|\Omega|}{|\mathcal{U}|}$ is not "too small".

- with "sufficiently many" samples, get an estimate of $|\Omega|$.

# efficient uniformly sampling

- *Uniform sampling :* Design a randomized algorithm $A$ :

- Given $\Omega$ (implicitly), output $A(\Omega)$ such that $\mathbf{Pr}(A(\Omega) = x) = 1/|\Omega|$ for each $x \in \Omega$.

- Eg: $\Omega$ = set of $(\Delta + 1)$-colorings of $G$.

- *Approximate sampling :* Design a rand. algorithm $A$ :

- Given $\Omega$ and $\delta$, output $A(\Omega)$ such that $d_1(A(.), \mathcal{U}) \leq 2\delta$.

- $A(.)$ distribution of the output of $A$ ; $\mathcal{U}$ is uniform.

- Required time bound is $poly(|\Omega|, \log \delta^{-1})$.

# approximate counting reduces to approximate sampling

- *Problem :* Given $G = (V, E)$, $k \geq \Delta + 2$ and $\epsilon > 0$,

- compute an $(1 \pm \epsilon)$ approximation to $\#col(G, k)$.

- *Approach :* $E = \{e_1, \ldots, e_m\}$. $E_i = \{e_1, \ldots, e_i\}$.

- $G_i = G(V, E_i)$. $G_0 = (V, \emptyset)$ and $G_m = G$.

- $\#col(G_0, k) = k^n$. Denote $N_i = \#col(G_i, k)$.

- $N_m = \frac{N_m}{N_{m-1}} \cdot \frac{N_{m-1}}{N_{m-2}} \cdot \ldots \cdot \frac{N_1}{N_0} \cdot k^n$.

- $(1 \pm \frac{\epsilon^2}{m})$ approximation for each ratio $\frac{N_i}{N_{i-1}}$ implies

- a $(1 \pm \epsilon)$ approximation for $N_m$.

- Required time bound is $poly(|V|, 1/\epsilon)$.

# approximate counting reduces to approximate sampling

- for every $i$, $N_i \leq N_{i-1}$ and $\frac{N_i}{N_{i-1}} \geq 1/2$.

- for every $i$, obtain "sufficiently many" and independent samples of a $k$-coloring of $G_{i-1}$ to get a $(1 \pm \frac{\epsilon^2}{m})$ approximation to $\frac{N_i}{N_{i-1}}$ in $poly(n, 1/\epsilon)$ time.

- Time complexity is $poly(|V|, 1/\epsilon)$.

- Applicable to problems like matchings, independent sets, etc.

- Generally, applicable to *self-reducible* problems.

# Markov chain based samplers

- Markov chain $M = (\Omega, P)$, $\Omega$ is finite.

- $P : \Omega \times \Omega \to [0, 1] : \forall x, \sum_y P(x, y) = 1$.

- $\forall t \geq 0, \ P^t(x, y) = \mathbf{Pr}(X_t = y | X_0 = x)$.

- $\pi_0$ - initial distribution at time $t = 0$.

- $\pi_t$ - distribution at time step $t$.

- $\pi_t = \pi_0 P^t$.

- $M$ is ergodic if there is some dist. $\pi : \Omega \to [0, 1]$ :
  $\forall x, y \in \Omega, \ Lim_{t \to \infty} P^t(x, y) = \pi(y)$.

## Markov chain based samplers

- $d_{TV}(\pi, \pi') = ||\pi - \pi'||_1/2$.

- Mixing time : $\tau(\epsilon) = \max_x \min\{t : d_{TV}(e_x P^t, \pi) \le \epsilon\}$.

- Desirable : $\tau(\epsilon) \le poly(n, \log \epsilon^{-1})$.

- several analytical tools available to

- rigorously bound the mixing time.

- coupling, conductance, cananonical paths, etc.

- set up a Markov chain, bound its mixing time and

- simulate the chain to get an approximately uniform sample.

# Glauber Dynamics

- MC for uniformly sampling $k$-colorings of
- $G$ where $k \geq \Delta(G) + 2$.
- $Q = \{1, \ldots, k\}$ and $X_t(S) = \{X_t(u) : u \in S\}$.
- **1.** Choose $u \in V$ uniformly at random.
- **2.** Choose $c \in Q \setminus X_t(N(u))$ u.a.r.
- **3.** $X_{t+1}(v) = X_t(v)$ if $v \neq u$ and $X_{t+1}(u) = c$.
- aperiodic and irreducible and hence ergodic.
- Can be shown to reach uniform distn asymptotically.
- also shown to mix in $O(n \log n)$ time for $k \geq 2\Delta + 1$.

# Conclusions

- Employing randomness leads to improved simplicity and improved efficiency in solving the problem.

- However, assumes the availability of a perfect source of independent and unbiased random bits.

- access to truly unbiased and independent sequence of random bits is expensive and should be considered as an expensive resource like time and space. One should aim to minimize the use of randomness to the extent possible.

- assumes efficient realizability of any rational bias. However, this assumption introduces error and increases the work and the required number of random bits.

- There are ways to reduce the randomness from several algorithms while maintaining the efficiency nearly the same.