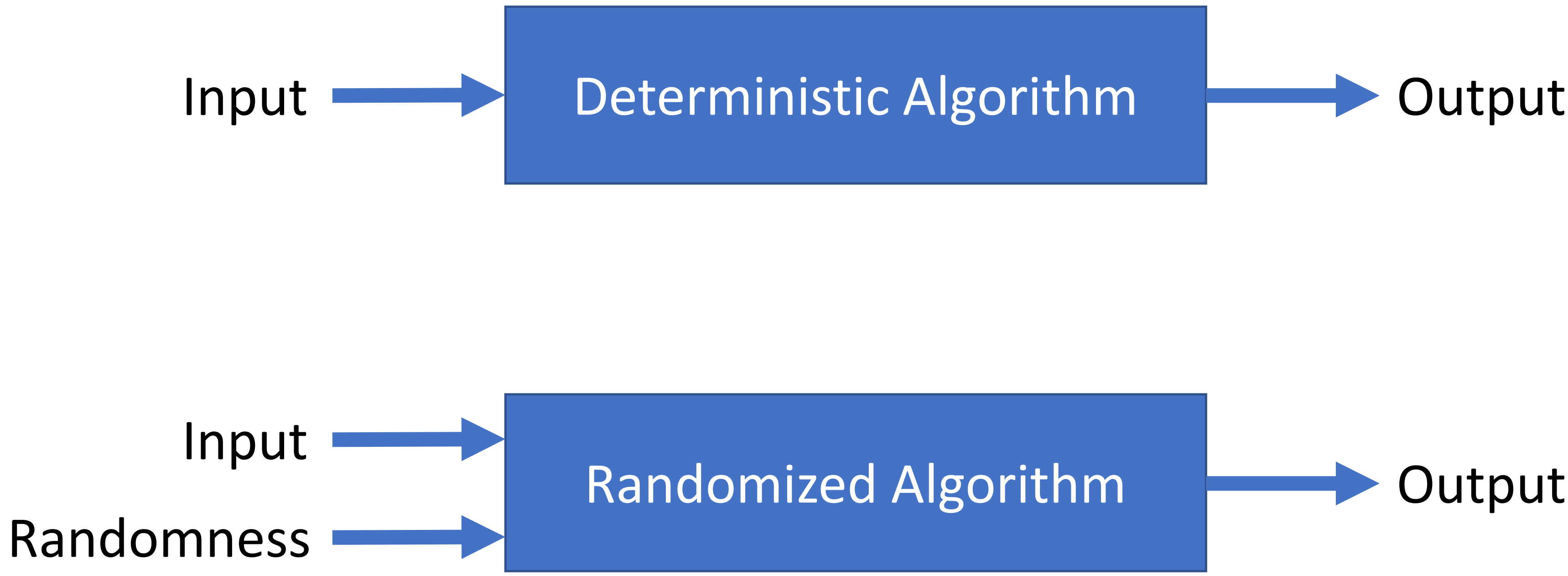
CSC373

Week 11: Randomized Algorithms

Randomized Algorithms



Randomized Algorithms

- Running time
  - Sometimes, we want the algorithm to *always* take a small amount of time
    - Regardless of both the input and the random coin flips
  - Sometimes, we want the algorithm to take a small amount of time *in expectation*
    - Expectation over random coin flips
    - Still regardless of the input

Randomized Algorithms

- Efficiency
  - We want the algorithm to return a solution that is, *in expectation,* close to the optimum according to the objective under consideration
    - Once again, the expectation is over random coin flips
    - We want this to hold for every input

Derandomization

- For some problems, it is easy to come up with a very simple randomized approximation algorithm

- Later, one can ask whether this algorithm can be "derandomized"
  - Informally, the randomized algorithm is making some random choices, and sometimes they turn out to be good
  - Can we make these "good" choices deterministically?

Recap: Probability Theory

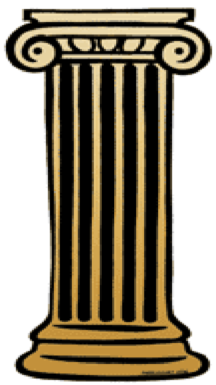
- Random variable X
  - Discrete
    - Takes value v_1 with probability p_1, v_2 w.p. p_2, …
    - Expected value $E[X] = p_1 \cdot v_1 + p_2 \cdot v_2 + \cdots$
    - Examples: the roll of a six-sided die (takes values 1 through 6 with probability $1/6$ each)
  - Continuous
    - Has a probability density function (pdf) f
    - Its integral is the cumulative density function (cdf) F
      - $F(x) = \Pr[X \le x]$
    - Expected value $E[X] = \int x\, f(x)\, dx$
    - Examples: normal distribution, exponential distribution, uniform distribution over $[0,1]$, …

Recap: Probability Theory

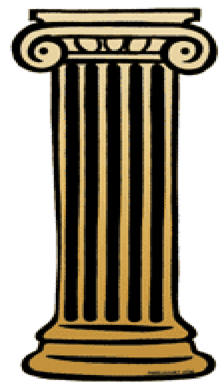
- Things you should be aware of…
  - Conditional probabilities
  - Independence among random variables
  - Conditional expectations
  - Moments of random variables
  - Standard discrete distributions: uniform over a finite set, Bernoulli, binomial, geometric, Poisson, …
  - Standard continuous distributions: uniform over intervals, Gaussian/normal, exponential, …

Three Pillars

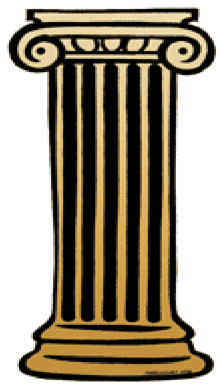
Linearity of Expectation



Union Bound



Chernoff Bound



- Deceptively simple, but incredibly powerful!
- Many many many many probabilistic results are just interesting applications of these three results

Three Pillars

- Linearity of expectation
  - $E[X + Y] = E[X] + E[Y]$
  - This does *not* require any independence assumptions about X and Y
  - E.g. if you want to find out how many people will attend your party on average, just ask each person the probability with which they will attend and add up
    - It does not matter that some of them are friends, and will either attend together or not attend together

Three Pillars

- Union bound
  - For any two events A and B, $\Pr[A \cup B] \leq \Pr[A] + \Pr[B]$
  - "Probability that at least one of the n events $A_1, \dots, A_n$ will occur is at most $\sum_i \Pr[A_i]$"
  - Typically, $A_1, \dots, A_n$ are "bad events"
    - You do not want any of them to occur
    - If you can individually bound $\Pr[A_i] \leq {}^{1}/_{2n}$ for each i, then probability that at least one them occurs $\leq 1/2$
    - So with probability $\geq {}^{1}/_{2}$, *none* of the bad events will occur

- Chernoff bound & Hoeffding's inequality
  - Read up!

Exact Max-k-SAT

Exact Max-k-SAT

- Problem (recall)
  - Input: An exact k-SAT formula $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where each clause C_i has exactly k literals, and a weight $w_i \geq 0$ of each clause C_i
  - Output: A truth assignment τ maximizing the number (or total weight) of clauses satisfied under τ

- Let us denote by $W(\tau)$ the total weight of clauses satisfied under τ

Exact Max-k-SAT

- Recall our local search
  - $N_d(\tau)$ = set of all truth assignments which can be obtained by changing the value of at most d variables in τ

- Result 1: Neighborhood $N_1(\tau) \Rightarrow {}^2/_3$-apx for Exact Max-2-SAT.
- Result 2: Neighborhood $N_1(\tau) \cup \tau^c \Rightarrow {}^3/_4$-apx for Exact Max-2-SAT.
- Result 3: Neighborhood $N_1(\tau)$ + oblivious local search $\Rightarrow {}^3/_4$-apx for Exact Max-2-SAT.

Exact Max-k-SAT

- Recall our local search
  - $N_d(\tau)$ = set of all truth assignments which can be obtained by changing the value of at most d variables in τ

- We claimed that ¾-apx for Exact Max-2-SAT can be generalized to $\frac{2^k-1}{2^k}$-apx for Exact Max-k-SAT
  - Algorithm becomes slightly more complicated

- What can we do with randomized algorithms?

Exact Max-k-SAT

- Recall:
  - We have a formula $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$
  - Variables = $x_1, \dots, x_n$, literals = variables or their negations
  - Each clause contains exactly k literals

- The most naïve randomized algorithm
  - Set each variable to TRUE with probability ½ and to FALSE with probability ½

- How good is this?

Exact Max-k-SAT

- Recall:
  - We have a formula $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$
  - Variables = $x_1, \ldots, x_n$, literals = variables or their negations
  - Each clause contains exactly k literals

- For each clause C_i:
  - $\Pr[C_i \text{ is not satisfied}] = 1/2^k$ (WHY?)
  - Hence, $\Pr[C_i \text{ is satisfied}] = (2^k - 1)/2^k$

Exact Max-k-SAT

- For each clause C_i:
  - $\Pr[C_i \text{ is not satisfied}] = 1/2^k$ (WHY?)
  - Hence, $\Pr[C_i \text{ is satisfied}] = (2^k - 1)/2^k$

- Let τ denote the random assignment
  - $E[W(\tau)] = \sum_{i=1}^{m} w_i \cdot \Pr[C_i \text{ is satisfied}]$

    (Which pillar did we just use?)

  - $E[W(\tau)] = \frac{2^k-1}{2^k} \cdot \sum_{i=1}^{m} w_i \geq \frac{2^k-1}{2^k} \cdot OPT$

Derandomization

- Can we derandomize this algorithm?
  - What are the choices made by the algorithm?
    - Setting the values of $x_1, x_2, \dots, x_n$
  - How do we know which set of choices is good?

- Idea:
  - Do not think about all the choices at once.
  - Think about them one by one.

Derandomization

- Say you want to *deterministically* make the right choice for x_1
  - Choices of $x_2, \ldots, x_n$ are still random

$$E[W(\tau)] = \Pr[x_1 = T] \cdot E[W(\tau) | x_1 = T] + \Pr[x_1 = F] \cdot E[W(\tau) | x_1 = F]$$
$$= {}^{1}/_{2} \cdot E[W(\tau) | x_1 = T] + {}^{1}/_{2} \cdot E[W(\tau) | x_1 = F]$$

  - This means at least one of $E[W(\tau) | x_1 = T]$ and $E[W(\tau) | x_1 = F]$ must be at least as much as $E[W(\tau)]$
    - Moreover, both quantities can be computed, so we can take the better of the two!
    - For now, forget about the running time…

Derandomization

- Once we have made the right choice for x_1 (say T), then we can apply the same logic to x_2
  - $$E[W(\tau)|x_1 = T] = {}^1\!/_2 \cdot E[W(\tau)|x_1 = T, x_2 = T] + {}^1\!/_2 \cdot E[W(\tau)|x_1 = T, x_2 = F]$$
  - And then we can pick the choice that leads to a better conditional expectation
- Derandomized Algorithm:
  - For $i = 1, \dots, n$
    - Let $z_i = T$ if $E[W(\tau)|x_1 = z_1, \dots, x_{i-1} = z_{i-1}, x_i = T] \geq E[W(\tau)|x_1 = z_1, \dots, x_{i-1} = z_{i-1}, x_i = F]$, and $z_i = F$ otherwise
    - Set $x_i = z_i$

Derandomization

- This is called *the method of conditional expectations*
  - If we're happy when making a choice at random, we should be at least as happy conditioned on at least one of the possible values of that choice

- Derandomized Algorithm:
  - For $i = 1, \dots, n$
    - Let $z_i = T$ if $E[W(\tau)|x_1 = z_1, \dots, x_{i-1} = z_{i-1}, x_i = T] \geq E[W(\tau)|x_1 = z_1, \dots, x_{i-1} = z_{i-1}, x_i = F]$, and $z_i = F$ otherwise
    - Set $x_i = z_i$

  - How do we compare the two conditional expectations?

Derandomization

- $E[W(\tau)|x_1 = z_1, \dots, x_{i-1} = z_{i-1}, x_i = T]$ vs $E[W(\tau)|x_1 = z_1, \dots, x_{i-1} = z_{i-1}, x_i = F]$
  - On both sides, we deterministically set $x_1, \dots, x_{i-1}$
  - Upon doing this, some clauses resolve to T/F
    - Their weights cancel out on both sides anyway
  - Some do not resolve, but also do not involve x_i or $\bar{x}_i$
    - Their expected weights also cancel out
  - For each clause C_r not of above two types
    - C_r contains x_i
      - Contribution to first term is w_r
      - Contribution to second term is $\frac{2^\ell - 1}{2^\ell} \cdot w_r$, where ℓ is the number of literals left in the clause after setting $x_1, \dots, x_{i-1}$ and $x_i = F$

Max-SAT

- Simple randomized algorithm
  - $\frac{2^k-1}{2^k}$ −approximation for Max-k-SAT
  - Max-3-SAT ⇒ $^7/_8$
    - [Håstad]: This is the best possible assuming P ≠ NP
  - Max-2-SAT ⇒ $^3/_4 = 0.75$
    - The best known approximation is 0.9401 using semi-definite programming and randomized rounding
  - Max-SAT ⇒ $^1/_2$
    - Max-SAT = no restriction on the number of literals in each clause
    - The best known approximation is 0.7968, also using semi-definite programming and randomized rounding

Max-SAT

- Better approximations for Max-SAT
  - Semi-definite programming is out of the scope
  - But we will see the simpler "LP + randomized rounding" approach that gives $1 - {}^1/_e \approx 0.6321$ approximation

- Max-SAT:
  - Input: $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where each clause C_i has weight $w_i \geq 0$ (and can have any number of literals)
  - Output: Truth assignment that approximately maximizes the number of clauses satisfied

LP Formulation of Max-SAT

- First, IP formulation:
  - Variables:
    - $y_1, \ldots, y_n \in \{0,1\}$
      - $y_i = 1$ iff variable $x_i = $ TRUE in Max-SAT
    - $z_1, \ldots, z_m \in \{0,1\}$
      - $z_j = 1$ iff clause C_j is satisfied in Max-SAT
    - Program:

      Maximize $\Sigma_j \, w_j \cdot z_j$
      s.t.
      $\Sigma_{x_i \in C_j} \, y_i + \Sigma_{\bar{x}_i \in C_j} \, (1 - y_i) \geq z_j \quad \forall j \in \{1, \ldots, m\}$
      $y_i, z_j \in \{0,1\} \quad \forall i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}$

LP Formulation of Max-SAT

- LP relaxation:
  - Variables:
    - $y_1, \ldots, y_n \in [0,1]$
      - $y_i = 1$ iff variable $x_i =$ TRUE in Max-SAT
    - $z_1, \ldots, z_m \in [0,1]$
      - $z_j = 1$ iff clause C_j is satisfied in Max-SAT
    - Program:

      Maximize $\Sigma_j \, w_j \cdot z_j$
      s.t.
      $\Sigma_{x_i \in C_j} \, y_i + \Sigma_{\bar{x}_i \in C_j} \, (1 - y_i) \geq z_j \quad \forall j \in \{1, \ldots, m\}$
      $y_i, z_j \in [0,1] \quad \forall i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}$

Randomized Rounding

- Randomized rounding
  - Find the optimal solution (y^*, z^*) of the LP
  - Compute a random IP solution $\hat{y}$ such that
    - Each $\hat{y}_i = 1$ with probability y_i^* and 0 with probability $1 - y_i^*$
    - Independently of other $\hat{y}_i$'s
    - The output of the algorithm is the corresponding truth assignment
  - What is $\Pr[C_j \text{ is satisfied}]$ if C_j has k literals?

$$1 - \Pi_{x_i \in C_j} (1 - y_i^*) \cdot \Pi_{\bar{x}_i \in C_j} (y_i^*)$$

$$\underbrace{\geq 1 - \left(\frac{\Sigma_{x_i \in C_j} (1 - y_i^*) + \Sigma_{\bar{x}_i \in C_j} (y_i^*)}{k} \right)^k}_{\text{AM-GM inequality}} \underbrace{\geq 1 - \left(\frac{k - z_j^*}{k} \right)^k}_{\text{LP constraint}}$$

Randomized Rounding

- Claim
  - $1 - \left(1 - \frac{z}{k}\right)^k \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right) \cdot z$ for all $z \in [0,1]$ and $k \in \mathbb{N}$

- Assuming the claim:

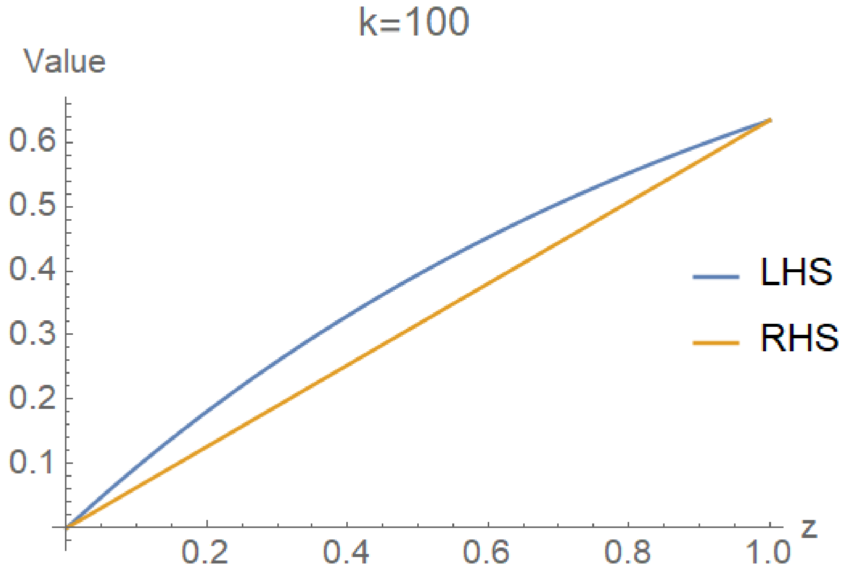
$$\Pr[C_j \text{ is satisfied}] \geq 1 - \left(\frac{k - z_j^*}{k}\right)^k \geq \underbrace{\left(1 - \left(1 - \frac{1}{k}\right)^k\right) \cdot z_j^* \geq \left(1 - \frac{1}{e}\right) \cdot z_j^*}_{\text{Standard inequality}}$$

- Hence,

$$\mathbb{E}[\text{\#weight of clauses satisfied}] \geq \left(1 - \frac{1}{\mathrm{e}}\right) \sum_j w_j \cdot z_j^* \geq \left(1 - \frac{1}{e}\right) \cdot OPT$$

Randomized Rounding

- Claim
  - $1-\left(1-\frac{z}{k}\right)^k \geq \left(1-\left(1-\frac{1}{k}\right)^k\right)\cdot z$ for all $z \in [0,1]$ and $k \in \mathbb{N}$
- Proof of claim:
  - True at $z=0$ and $z=1$ (same quantity on both sides)
  - For $0 \leq z \leq 1$:
    - LHS is a convex function
    - RHS is a linear function
    - Hence, LHS $\geq$ RHS ■



Improving Max-SAT Apx

- Claim without proof:
  - Running both "LP + randomized rounding" and "naïve randomized algorithm", and returning the best of the two solutions gives $3/4 = 0.75$ approximation!
  - This algorithm can be derandomized.
  - Recall:
    - "naïve randomized" = independently set each variable to TRUE/FALSE with probability 0.5 each, which only gives $1/2 = 0.5$ approximation by itself

Back to 2-SAT

- Max-2-SAT is NP-hard (we didn't prove this!)
- But 2-SAT can be efficiently solved
  - "Given a 2-CNF formula, check whether *all* clauses can be satisfied simultaneously."
- Algorithm:
  - Eliminate all unit clauses, set the corresponding literals.
  - Create a graph with $2n$ literals as vertices.
  - For every clause $(x \vee y)$, add two edges: $\bar{x} \rightarrow y$ and $\bar{y} \rightarrow x$.
    - $u \rightarrow v$ means if u is true, v must be true.
  - Formula is satisfiable iff no path from x to $\bar{x}$ or $\bar{x}$ to x for any x
  - Solve $s - t$ connectivity problem in polynomial time

Random Walk + 2-SAT

- Here's a cute randomized algorithm by Papadimitriou [1991]

- **Algorithm:**
  - Start with an arbitrary assignment.
  - While there is an unsatisfied clause $C = (x \vee y)$
    - Pick one of the two literals with equal probability.
    - Flip the variable value so that C is satisfied.

- But, but, this can hurt other clauses?

Random Walk + 2-SAT

- Theorem:
  - If there is a satisfying assignment τ^*, then the expected time to reach some satisfying assignment is at most $O(n^2)$.
- Proof:
  - Fix τ^*. Let τ_0 be the starting assignment. Let τ_i be the assignment after i iterations.
  - Consider the "hamming distance" d_i between τ_i and τ^*
    - Number of coordinates in which the two differ
    - $d_i \in \{0,1,\ldots,n\}$.
  - We want to show that in expectation, we will hit $d_i = 0$ in $2n^2$ iterations, unless the algorithm stops before that.

Random Walk + 2-SAT

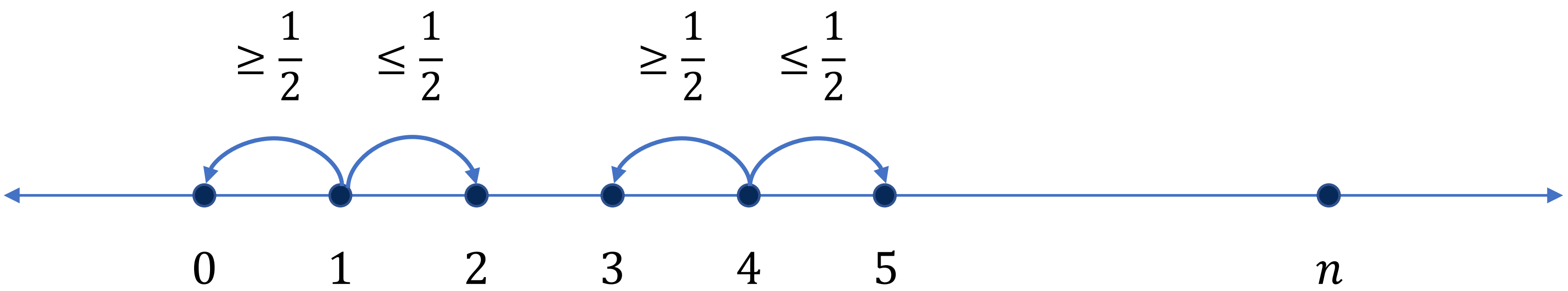
- Observation: $d_{i+1} = d_i - 1$ or $d_{i+1} = d_i + 1$
  - Because we change one variable in each iteration.
- Claim: $\Pr[d_{i+1} = d_i - 1] \geq 1/2$
- Proof:
  - Iteration i considers an unsatisfied clause $C = (x \vee y)$
  - τ^* satisfies at least one of x or y, while τ_i satisfies neither
  - Because we pick a literal randomly, w.p. at least ½ we pick one where τ_i and τ^* differ, and decrease distance.
  - Q: Why did we need an unsatisfied clause? What if we pick one of n variables randomly, and flip it?

Random Walk 2-SAT

- Answer:
  - We want the distance to decrease with probability at least $\frac{1}{2}$ no matter how close or far we are from τ^*.
  - If we are already close, choosing a variable at random will likely choose one where τ and τ^* already match.
  - Flipping this variable will increase the distance with high probability.
  - An unsatisfied clause narrows it down to two variables s.t. τ and τ^* differ on at least one of them

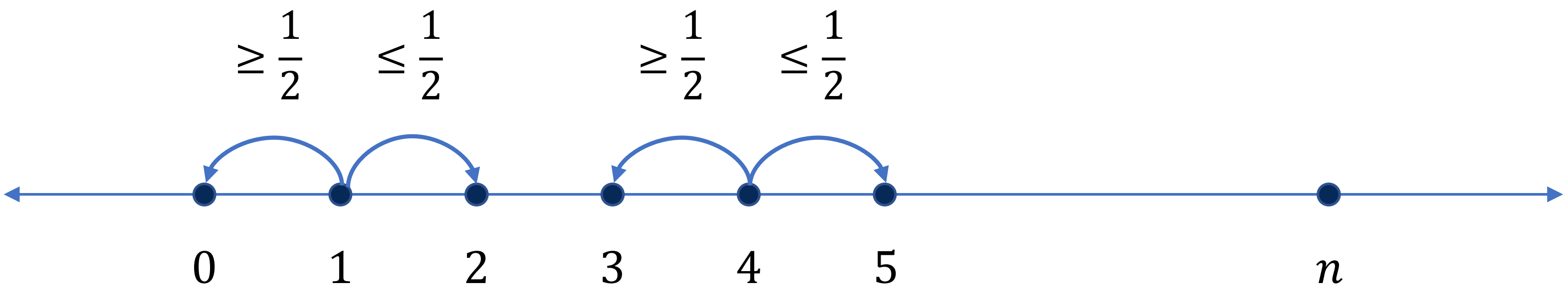
Random Walk + 2-SAT

- Observation: $d_{i+1} = d_i - 1$ or $d_{i+1} = d_i + 1$
- Claim: $\Pr[d_{i+1} = d_i - 1] \geq 1/2$



- How does this help?

Random Walk + 2-SAT



- How does this help?
  - Can view this as Markov chain and use hitting time results
  - But let's prove it with elementary methods.
  - $T_{i+1,i}$ = expected time to go from $i + 1$ to i
    - $T_{i+1,i} \leq \left(\frac{1}{2}\right) * 1 + \left(\frac{1}{2}\right) * T_{i+2,i} \leq \frac{1}{2} + \left(\frac{1}{2}\right) * \left(T_{i+2,i+1} + T_{i+1,i}\right)$
    - Thus, $T_{i+1,i} \leq 1 + T_{i+2,i+1} \rightarrow T_{i+1,i} = O(n)$
    - $T_{n,0} \leq T_{n,n-1} + \cdots + T_{1,0} = O(n^2)$

Random Walk + 2-SAT

- Can view this algorithm as a "drunken local search"
  - We are searching the local neighborhood
  - But we don't ensure that we necessarily improve.
  - We just ensure that in expectation, we aren't hurt.
  - Hope to reach a feasible solution in polynomial time
- Schöning extended this technique to k-SAT
  - Schöning's algorithm no longer runs in polynomial time, but this is okay because k-SAT is NP-hard
  - It still improves upon the naïve 2^n
  - Later derandomized by Moser and Scheder [2011]

Schöning's Algorithm for k-SAT

- Algorithm:
  - Choose a random assignment τ.
  - Repeat $3n$ times ($n =$ #variables)
    - If τ satisfies the CNF, stop.
    - Else, pick an arbitrary unsatisfied clause, and flip a random literal in the clause.

Schöning's Algorithm

- Randomized algorithm with one-sided error
  - If the CNF is satisfiable, it finds an assignment with probability at least $\left(\frac{1}{2} \cdot \frac{k}{k-1}\right)^n$
  - If the CNF is unsatisfiable, it surely does not find an assignment.

- Expected # times we need to repeat = $\left(2\left(1 - \frac{1}{k}\right)\right)^n$
  - For $k = 3$, this gives $O(1.3333^n)$
  - For $k = 4$, this gives $O(1.5^n)$

Best Known Results

- 3-SAT

- Deterministic
  - Derandomized Schöning's algorithm: $O(1.3333^n)$
  - Best known: $O(1.3303^n)$ [HSSW]
    - If there is a unique satisfying assignment: $O(1.3071^n)$ [PPSZ]

- Randomized
  - Nothing better known without one-sided error
  - With one-sided error, best known is $O(1.30704^n)$ [Modified PPSZ]

Random Walk + 2-SAT

- Random walks are not only of theoretical interest
  - WalkSAT is a practical SAT algorithm
  - At each iteration, pick an unsatisfied clause *at random*
  - Pick a variable in the unsatisfied clause to flip:
    - With some probability, pick at random.
    - With the remaining probability, pick one that will make the fewest previously satisfied clauses unsatisfied.
  - Restart a few times (avoids being stuck in local minima)
- Faster than “intelligent local search” (GSAT)
  - Flip the variable that satisfies most clauses

Random Walks on Graphs

- Aleliunas et al. [1979]
  - Let G be a connected undirected graph. Then a random walk starting from any vertex will cover the entire graph (visit each vertex at least once) in $O(mn)$ steps.
- Also care about limiting probability distribution
  - In the limit, the random walk with spend $\frac{d_i}{2m}$ fraction of the time on vertex with degree d_i
- Markov chains
  - Generalize to directed (possibly infinite) graphs with unequal edge probabilities

Randomization for Sublinear Running Time

Sublinear Running Time

- Given an input of length n, we want an algorithm that runs in time $o(n)$
  - $o(n)$ examples: $\log n$, $\sqrt{n}, n^{0.999}, \frac{n}{\log n}, \ldots$
  - The algorithm doesn't even get to read the full input!
  - There are four possibilities:
    - Exact vs inexact: whether the algorithm always returns the correct/optimal solution or only does so with high probability (or gives some approximation)
    - Worst-case versus expected running time: whether the algorithm always takes $o(n)$ time or only does so in expectation (but still on every instance)

Exact algorithms, expected sublinear time

Searching in Sorted List

- Input: A sorted doubly linked list with n elements.
  - Imagine you have an array A with $O(1)$ access to $A[i]$
  - $A[i]$ is a tuple (x_i, p_i, n_i)
    - Value, index of previous element, index of next element.
  - Sorted: $x_{p_i} \leq x_i \leq x_{n_i}$
- Task: Given x, check if there exists i s.t. $x = x_i$
- Goal: We will give a randomized + exact algorithm with expected running time $O(\sqrt{n})$!

Searching in Sorted List

- Motivation:
  - Often we deal with large datasets that are stored in a large file on disk, or possibly broken into multiple files
  - Creating a new, sorted version of the dataset is expensive
  - It is often preferred to "implicitly sort" the data by simply adding previous-next pointers along with each element
  - Would like algorithms that can operate on such implicitly sorted versions and yet achieve sublinear running time
    - Just like binary search achieves for an explicitly sorted array

Searching in Sorted List

Algorithm:

- Select $\sqrt{n}$ random indices R
- Access x_j for each $j \in R$
- Find the nearest $x_j : j \in R$ on each side of x
  - $p \in R$ such that $x_p = \max\{x_j : x_j \le x, j \in R\}$
  - $q \in R$ such that $x_q = \min\{x_j : x_j > x, j \in R\}$
  - One of the two must exist (WHY?).
- If p exists, start at $A[p]$, and keep going next until you discover x, or you reach $A[q]$ or end of list.
- If q exists, start at $A[q]$, and keep going back until you discover x, or you reach $A[p]$ or beginning of list.

Searching in Sorted List

- Analysis:
  - Take arbitrary value x. Take the minimum value x_i in the list that is at least x. The algorithm is searching for x_i.
  - The algorithm throws $\sqrt{n}$ random "darts" on the list.
  - Chernoff bounds: the probability that there is no dart in $c\sqrt{n}$ elements to the left (resp. right) of x_i is $2^{-\Omega(c)}$.
    - Exercise!
  - So, the expected distance of x_i to the dart on its left (and its right) is $O(\sqrt{n})$.
  - The algorithm finds these two darts in $O(\sqrt{n})$ time, and uses $O(\sqrt{n})$ search to locate x_i.

Searching in Sorted List

- Note:
  - We don't *really* require the list to be doubly linked. Just "next" pointer suffices if we have a pointer to the first element of the list (a.k.a. "anchored list").

- This algorithm is optimal!

- Theorem: No algorithm that always returns the correct answer can run in $o(\sqrt{n})$ expected time.
  - Can be proved using Yao's minimax principle
  - Beyond the scope of the course, but this is a fundamental result with wide-ranging applications

Sublinear Geometric Algorithms

- Chazelle, Liu, and Magen [2003] proved the $\Theta(\sqrt{n})$ bound for searching in a sorted linked list
  - Their main focus was to generalize these ideas to come up with sublinear algorithms for geometric problems
  - Polygon intersection: Given two convex polyhedra, check if they intersect.
  - Point location: Given a Delaunay triangulation (or Voronoi diagram) and a point, find the cell in which the point lies.
  - They provided optimal $O(\sqrt{n})$ algorithms for both these problems.

Inexact algorithms, expected sublinear time

Estimating Avg Degree in Graph

- Input: Graph G with n vertices, and access to an oracle that returns the degree of a queried vertex in $O(1)$ time.

- Goal: $(2+\epsilon)$-approximation in expected time $O\left(\epsilon^{-O(1)}\sqrt{n}\right)$
  - ϵ is constant $\Rightarrow$ sublinear in input size n

Estimating Avg Degree in Graph

- Wait!
  - Isn't this equivalent to "given an array of n numbers between 1 and $n-1$, estimate their average"?
  - No! That requires $\Omega(n)$ time for constant approximation!
    - Consider an instance with constantly many $n-1$'s, and all other 1's: you may not discover any $n-1$ until you query $\Omega(n)$ numbers
  - Why are degree sequences more special?
    - Erdős–Gallai theorem: $d_1 \geq \cdots \geq d_n$ is a degree sequence iff their sum is even and $\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} d_i$.
    - Intuitively, we will sample $O(\sqrt{n})$ vertices
      - We may not discover the few high degree vertices, but we'll find their neighbors, and thus account for their edges anyway!

Estimating Avg Degree in Graph

- Algorithm:
  - Take $8/_{\epsilon}$ random subsets $S_i \subseteq V$ with $|S_i| = s$
  - Compute the average degree d_{S_i} in each S_i.
  - Return $\widehat{d} = \min_i d_{S_i}$

- Analysis beyond the scope of this course
  - But doesn't use anything other than Hoeffding's inequality, Markov's inequality, linearity of expectation, and union bound