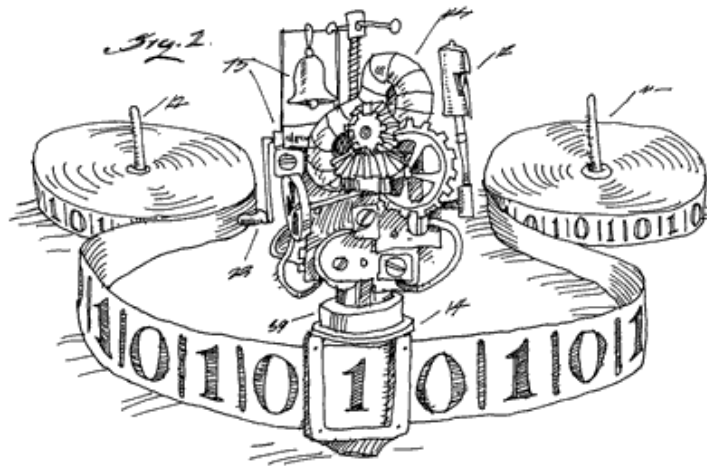


EECS 376: Foundations of Computer Science





Lecture 20 - Randomness in Computation



Randomized Algorithms



Techniques and Paradigms in this Course

- Divide-and-conquer, greed, dynamic programming, **the power of randomness**

- Computability 
- NP-completeness and approximation algorithms

- Cryptography


What do we mean by “randomized algorithm”?

Goal: Use randomization to get faster and/or more accurate algorithms (still for worst-case input).

We are **NOT** considering randomly chosen inputs.

E.g. Quicksort (next lecture):

“The expected running time is $O(n \log n)$ ” means:

“**For ANY given array**, in expectation (over the random choices made by the algorithm) the running time is $O(n \log n)$ ”

What do we mean by “randomized algorithm”?

Goal: Use randomization to get faster and/or more accurate algorithms (still for worst-case input).

We are **NOT** considering randomly chosen inputs.

In general, we will make statements like:

[In expectation] or [with probability at least p]

+

[runtime of algorithm is $O(f(n))$] or [algorithm returns correct answer]

Always imagine “for ANY given input” precedes such statements.

Heisenberg Uncertainty Principle

- The **Heisenberg Uncertainty Principle** is a fundamental theory in **quantum mechanics**, introduced by German physicist **Werner Heisenberg** in 1927.
- It states that it is **impossible** to simultaneously know both the exact **position** and exact **momentum** of a particle.
- In other words, the more precisely one property is measured, the less precisely the other can be controlled or determined. This is not due to any technological limitations, but rather a result of the **inherent** properties of **quantum systems**.

Heisenberg Uncertainty Principle

$$\sigma_x \cdot \sigma_p \leq \frac{\hbar}{2}$$

- σ_x is the standard deviation of the position
- σ_p is the standard deviation of the momentum
- \hbar is the reduced Planck constant, which is approximately $1.0545718 \times 10^{-34}$ joule-seconds (J·s)

The Planck constant

- The Planck constant, symbolized as h , is a fundamental physical constant that is of paramount importance in quantum mechanics.
- It relates the energy of a photon to its frequency through the Planck-Einstein relation: $E = h \times f$, where E is the energy, f is the frequency, and h is the Planck constant.
- Max Planck originally introduced this constant in 1900 to explain black-body radiation, and it has since become a cornerstone of quantum physics.
- The reduced Planck constant, $\hbar = \frac{h}{2\pi}$.

Where do random numbers come from?

In this class, we assume access to a perfect random number generator.

The app contacts a remote quantum device in Geneva.



Universe Splitter 4+

Quantum Decision-Maker

Aerfish LLC

Designed for iPad

#15 in Entertainment

★★★★★ 4.6 • 1.2K Ratings

\$1.99

[View in Mac App Store](#)

In practice, we use “good enough” random number generators.

If quantum physics produces true randomness,
we can also use truly random numbers.



(Humans are generally not good at generating random numbers).



Why would you ever use a randomized algorithm?

...after all, there's no randomness in the problem statement

- Sometimes randomization is **provably necessary** →
- Sometimes we **don't know an efficient deterministic algorithm** (even though one may exist)
- Sometimes randomization is not necessary but leads to **simpler** or **faster-in-practice** algorithms



battleship!

- However, for some applications we have to be careful: don't want an autonomous vehicle that fails 0.5% of the time

Battleship

When **deterministic** algorithms
are **provably** much worse than
randomized algorithm

The Battleship Problem

Input: A hidden battleship board. When you guess a cell, its status (empty/occupied) is revealed.

	1	2	3	4	5	6	7	8	9	10
A	■							■		
B								■		
C		■						■		
D								■		
E				■				■		
F				■				■		■
G				■				■		■
H								■		■
I						■				■
J		■								

Output: Any cell occupied by a ship.

I prefer
battleship



1-Dimensional Battleship Problem

Input: Hidden 0/1 array **A** of length **n**, with exactly **$n/4$** 1's.
When you guess a cell, its value is revealed.

0	0	1	0	0	0	1	1	0	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Output: Any i such that $A[i] = 1$.

What algorithm would you use?

1-Dimensional Battleship Problem

Input: Hidden 0/1 array **A** of length **n**, with exactly **$n/4$** 1's.
When you guess a cell, its value is revealed.

0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Output: Any i such that $A[i] = 1$.

Linear scan algorithm:
for $i = 1 \dots n$:
 if $A[i] = 1$: return i

Worst-case number of guesses $\geq (3/4)n$

1-Dimensional Battleship Problem

Input: Hidden 0/1 array **A** of length **n**, with exactly **$n/4$** 1's.
When you guess a cell, its value is revealed.

1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Output: Any i such that $A[i] = 1$.

```
Backwards linear scan:  
for  $i = n \dots 1$ :  
    if  $A[i] = 1$ : return  $i$ 
```

Worst-case number of guesses $\geq (3/4)n$

1-Dimensional Battleship Problem

Input: Hidden 0/1 array **A** of length **n**, with exactly **$n/4$** 1's.
When you guess a cell, its value is revealed.

0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Output: Any i such that $A[i] = 1$.

Alternating linear scan:
for $i = 1, n, 2, n-1, 3, n-2, \dots$:
if $A[i] = 1$: return i

Worst-case number of guesses $\geq (3/4)n$

1-Dimensional Battleship Problem

Input: Hidden 0/1 array **A** of length **n**, with exactly **$n/4$** 1's.
When you guess a cell, its value is revealed.

0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Output: Any i such that $A[i] = 1$.

For **any** deterministic algorithm,
there **exists** an input
such the number of guesses is $\geq (3/4)n$.

1-Dimensional Battleship Problem

Input: Hidden 0/1 array **A** of length **n**, with exactly **$n/4$** 1's.
When you guess a cell, its value is revealed.

0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Output: Any i such that $A[i] = 1$.

Random algorithm:

while true:

$i = \text{random integer in } [1 \dots n]$

if $A[i] = 1$: return i

For any given input, expected number of guesses = 4
(**Exercise:** Formally prove this after class)

Rock-Paper-Scissor

When **deterministic** algorithms
are **provably** much worse than
randomized algorithm

Concept: Independence

Let's look at another toy example to review some concepts in probability:

Rock-Paper-Scissors

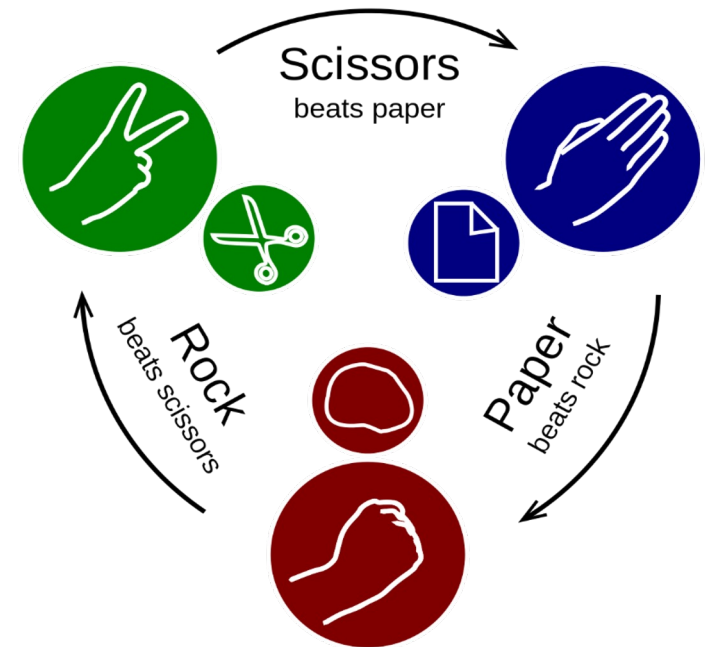
Goal: Minimize our probability of losing in a (best-of-one) game of rock-paper-scissors against an opponent who knows our strategy.

With any deterministic strategy, we lose (if opponent is logical). **Why?**

Strategy: Play uniformly at random.

Intuition: We lose with probability $1/3$.

Let's prove it!



Rock-Paper-Scissors

Terminology Review

The **sample space** (set of all possible outcomes)

	(R, R)	(R, P)	(R, S)	we play rock
they play rock	(P, R)	(P, P)	(P, S)	
	(S, R)	(S, P)	(S, S)	

Each outcome is of the form (what we play, what they play)

An **event** (subset of outcomes)

Each outcome has a **probability** and all of the probabilities add to 1.

The **probability** of an **event** is the sum of the probabilities of the outcomes in that event.

Rock-Paper-Scissors

The *sample space* (set of all possible outcomes)

probability they play rock depends on their strategy	(R, R)	(R, P)	(R, S)	probability we play rock = $\frac{1}{3}$
	(P, R)	(P, P)	(P, S)	
	(S, R)	(S, P)	(S, S)	

Each outcome is of the form (what we play, what they play)

We can't calculate the probability of (R, R) or any other outcome since we don't know their strategy.

But we can still prove we lose with probability $\frac{1}{3}$ using *independence*.

203 Review: Independence

Intuitively: Two events are *independent* if the occurrence of one does not affect the probability of the other occurring.

Formal Definition: Events **A** and **B** are *independent* if

$$\Pr[\mathbf{A} \cap \mathbf{B}] = \Pr[\mathbf{A}] \cdot \Pr[\mathbf{B}].$$

(we won't prove the equivalence of the intuition and the formal defn)

E.g. we flip two fair coins:

A is the event that coin 1 is heads,

B is the event that coin 2 is heads.

A and B are independent events and the probability both are

heads is: $\Pr[\mathbf{A} \cap \mathbf{B}] = \Pr[\mathbf{A}] \cdot \Pr[\mathbf{B}] = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$.

Notice that our random rock-paper-scissors choice is *independent* of our opponent's choice

Rock-Paper-Scissors

Analysis

Claim. *Against any opponent, by picking rock, paper, or scissors uniformly at random, we lose with probability $\frac{1}{3}$.*

Event A_x = we pick x ; Event B_y = they pick y

The probability that we **lose** is:

$$\Pr(A_R \cap B_P) + \Pr(A_P \cap B_S) + \Pr(A_S \cap B_R)$$

Rock-Paper-Scissors

Analysis

Claim. *Against any opponent, by picking rock, paper, or scissors uniformly at random, we lose with probability $\frac{1}{3}$.*

Event A_x = we pick x ; Event B_y = they pick y

The probability that we **lose** is:

$$\Pr(A_R \cap B_P) + \Pr(A_P \cap B_S) + \Pr(A_S \cap B_R)$$

$$= \frac{1}{3} (\Pr(B_P) + \Pr(B_S) + \Pr(B_R)) \text{ (independence)}$$

$$= \frac{1}{3} \text{ (since they either pick R, P, or S; covers all outcomes)}$$

$\frac{1}{3}$ win, $\frac{1}{3}$ draw, $\frac{1}{3}$ lose.

Max-3SAT

Concepts:

Random Variable, Expected values,
Indicator, Linearity of Expectation

Max-3SAT

3SAT is NP-hard but what if we try the easier goal of satisfying as many clauses as we can?

Input: 3CNF formula where *each clause has 3 distinct variables*.

E.g. $(x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z)$
 $\wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z)$

Our goal: Output an assignment that satisfies at least $\frac{7}{8}$ of the clauses.

Wait, does such an assignment always exist? **Yes!** (we will show)

What about for a fraction bigger than $\frac{7}{8}$? **No!** (above example)

Where did the
number $\frac{7}{8}$
come from?



We could try coming up with a deterministic algorithm...but that sounds complicated.

Consider a very simple **randomized** algorithm:

Pick an assignment uniformly at random!

We will show: For any 3CNF formula,
the ***expected value*** of the fraction of satisfied clauses is $7/8$.

But first let's review the concept of expected value...

203 Review: Random Variables

A **random variable** is a quantity determined by the outcome of *a random experiment*.

E.g. Let **N** be the number of satisfied clauses of 3CNF formula ϕ when we *assign variables T/F independently and uniformly at random*.

$$\phi = \underbrace{(x \vee y \vee z)}_{c1} \wedge \underbrace{(\neg x \vee y \vee z)}_{c2} \wedge \overline{\underbrace{(\neg x \vee y \vee z)}_{c3}} \quad \text{(c2 and c3 are the same but that's ok)}$$

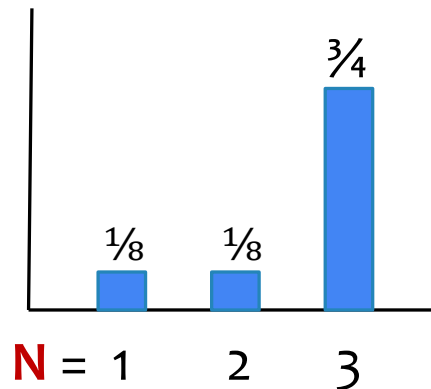
	x y z			x y z ...				
Outcome	F F F	F F T	F T F	F T T	T F F	T F T	T T F	T T T
	c1 c2 c3			c1 c2 c3 ...				
Sat?	N Y Y	Y Y Y	Y Y Y	Y Y Y	Y N N	Y Y Y	Y Y Y	Y Y Y
N	2	3	3	3	1	3	3	3

203 Review: Distribution of a Random Variable

The **probability** that a random variable equals some fixed value **v** is the sum of the probabilities of all outcomes that result in value **v**.

N	2	3	3	3	1	3	3	3
Pr[outcome]:	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$

$$\Pr[\mathbf{N}=1] = \frac{1}{8}, \quad \Pr[\mathbf{N}=2] = \frac{1}{8}, \quad \Pr[\mathbf{N}=3] = \frac{3}{4}$$



203 Review: Expected Value of a Random Variable

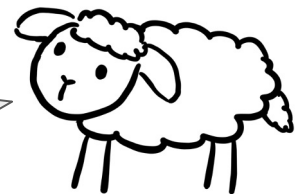
The **expected value** of a random variable N is the weighted average of its values:

$$\mathbb{E}[N] = \sum_v v \cdot \Pr[N = v]$$

E.g. Since $\Pr[\mathbf{N}=1] = 1/8$, $\Pr[\mathbf{N}=2] = 1/8$, $\Pr[\mathbf{N}=3] = 3/4$,
 $\mathbb{E}[\mathbf{N}] = 1 \cdot 1/8 + 2 \cdot 1/8 + 3 \cdot 3/4 = 2.625$ clauses (which is a $7/8$ fraction).

“In expectation, a random assignment satisfies $7/8^{\text{ths}}$ of the clauses of ϕ .”

The expected value
doesn't have to be
possible to achieve



We proved: “*In expectation*, a random assignment satisfies $\frac{7}{8}$ ^{ths} of the clauses of ϕ .”

But ϕ was one specific 3CNF formula, and we want to prove this for **all** 3CNF formulas (where each clause has 3 distinct variables).

But how? The clauses are not independent of each other!

A very useful trick: indicator variables + linearity of expectation



203 Review: 0/1 Indicator Random Variables

An **indicator** random variable X has 2 possible outcomes: 0 and 1.

Expected value of an indicator r.v.: $E[X] = \Pr[X=1]$. Why?

E.g. For a 3CNF formula $C_1 \wedge C_2 \wedge \dots \wedge C_m$, with a random assignment of variables, let N_i be the **indicator** random variable:

$$\begin{cases} 1 & \text{if clause } C_i \text{ is satisfied} \\ 0 & \text{otherwise} \end{cases}$$

Observation: The number of clauses satisfied by the assignment is $N = N_1 + N_2 + \dots + N_m$.

Review: Linearity of Expectation

Linearity of Expectation: For any (not necessarily independent!) random variables N_i :

$$\mathbb{E}\left[\sum_i N_i\right] = \sum_i \mathbb{E}[N_i].$$

For example:

Let \mathbf{X} be the outcome of rolling a 6-sided dice (expected value 3.5)

Let \mathbf{Y} be the outcome of rolling a 12-sided dice (expected value 6.5)

Then the expected sum $\mathbb{E}[\mathbf{X}+\mathbf{Y}]$ equals...

$$\mathbb{E}[\mathbf{X}+\mathbf{Y}] = \mathbb{E}[\mathbf{X}] + \mathbb{E}[\mathbf{Y}] = 3.5 + 6.5 = 10.$$



Example to demonstrate the magic of indicator rv's + linearity of expectation



Suppose I have a name tag for all n people in this class.
I hand them out randomly.

Compute: the expected number of people who got their own tag.

Let X_i be an *indicator random variable* for whether person i receives their own name tag. For all i , $E[X_i] = \Pr[X_i = 1] = 1/n$.

Observe that $\sum_{i=1}^n X_i$ is the number of people who receive their own name tag. So our goal is to calculate $E[\sum_{i=1}^n X_i]$.

By *linearity of expectation*, $E[\sum_{i=1}^n X_i] = \sum_{i=1}^n E[X_i] = \sum_{i=1}^n 1/n = 1$.

The answer doesn't even depend on the number of people?!



Now we're ready to do what we set out to do...

Given a 3CNF formula $\phi = C_1 \wedge C_2 \wedge \dots \wedge C_m$ where each clause has 3 distinct variables, we pick a random T/F assignment of the variables.

Your task: Calculate the $E[N]$.

N is the number of clauses satisfied.

N_i is an indicator random variable for whether clause C_i is satisfied.

Now we're ready to do what we set out to do...

Given a 3CNF formula $\phi = C_1 \wedge C_2 \wedge \dots \wedge C_m$ where each clause has 3 distinct variables, we pick a random T/F assignment of the variables.

Your task: Calculate the $E[N]$.

N is the number of clauses satisfied.

N_i is an indicator random variable for whether clause C_i is satisfied.

What is $\Pr[C_i \text{ not satisfied}]$?

- $\Pr[3 \text{ literals in } C_i \text{ set to false}] = \left(\frac{1}{2}\right)^3 = \frac{1}{8}$ by independence.

What is $E[N_i]$?

- $E[N_i] = \Pr[C_i \text{ satisfied}] = 1 - \frac{1}{8} = \frac{7}{8}$

What is $E[N]$?

- $E[N] = E[\sum_{i=1}^m N_i] = \sum_{i=1}^m E[N_i] = \frac{7}{8} m$ by linearity of expectation.

Max-3SAT

Algo: Pick an assignment uniformly at random!

We showed: For any 3CNF formula,
the **expected value** of the fraction of satisfied clauses is $\frac{7}{8}$.

This is a statement of a random process.

Can we guarantee something deterministically?

Expectation \Rightarrow Existence

Fact: For any r.v. \mathbf{X} there *exists* an outcome $\geq E[\mathbf{X}]$.

Fact: For any r.v. \mathbf{X} there *exists* an outcome $\leq E[\mathbf{X}]$.

Thus, for **every** 3CNF (where each clause has 3 distinct variables), there **exists** an assignment satisfying at least $\frac{7}{8}$ of the clauses.

Max-3SAT

Algo: Pick an assignment uniformly at random!

We showed: For any 3CNF formula,
the **expected value** of the fraction of satisfied clauses is $\frac{7}{8}$.

Can we say that a random assignment satisfies $\frac{3}{4}$ of clauses with large probability?

Markov Bound

Markov's Inequality

“A non-negative random variable X is rarely way bigger than its expectation.”



Markov's Inequality: If X is a non-negative random variable and $a > 0$, then $\Pr[X \geq a] \leq \mathbb{E}[X]/a$.

Proof: $\mathbb{E}[X] \geq a\Pr[X \geq a]$ by definition of expectation.

Divide by a . $\mathbb{E}[X] = \sum_{i=1}^{\infty} a_i \Pr[X = a_i] \geq \underbrace{a \Pr[X \geq a]}_{(a \text{ 为 } \mathbb{E}[X] \text{ 的一个})}$

Applying Markov

Want to say: a random assignment satisfies $\frac{3}{4}$ of clauses with large probability.

Let N be the number of satisfied clauses.

Want: lower bound $\Pr[N \geq \frac{3m}{4}]$.

Can we use Markov?

$$\text{By Markov, } \Pr[N \geq 3m/4] \leq \frac{E[N]}{3m/4} = \frac{7m/8}{3m/4} = 1.16 \dots$$

We get a meaningless upper bound instead of lower bound.
What should we do?

Applying Markov

Want to say: a random assignment satisfies $\frac{3}{4}$ of clauses with large probability.

Let N' the number of **unsatisfied** clauses.

$$\text{By Markov, } \Pr[N' \geq m/4] \leq \frac{E[N']}{m/4} = \frac{m/8}{m/4} = 0.5.$$

That is, $N' \geq m/4$ with probability **at most 0.5**.

So, $N \geq 3m/4$ with probability **at least 0.5**.

$$\Pr[X > v] \geq \frac{E[X] - v}{b - v} = \frac{\frac{7}{8}m - \frac{3}{4}m}{m - \frac{3}{4}m} = 0.5m$$

Boosting Success Probability

Very useful trick

Simple trick: repeat!

We can get an assignment satisfying **at least $\frac{3}{4}$** of clauses with probability **at least $\frac{1}{2}$** . (Just pick a random assignment)

Can we boost this probability to 0.999? Easy!

Algo:

- Sample 100 random assignments independently.
- Take the best one.

$$\Pr[\text{all } k \text{ assignments satisfies less than } \frac{3}{4} \text{ of clauses}] \leq \left(\frac{1}{2}\right)^{100}$$

Conclude: One of the assignments satisfies **at least $\frac{3}{4}$** of clauses with probability at least $1 - (1/2)^{100}$

Derandomization

via expectation maximization

Derandomizing the Algorithm

using a technique called “*expectation maximization*”

Theorem: There is an efficient *deterministic* algorithm that outputs an assignment satisfying $7/8$ ths of the clauses.

Let z_1, \dots, z_n be the variables in the formula.

Suppose I already chose an assignment for variables z_1, \dots, z_i .

Choose $z_{i+1} = \mathbf{T}$ or \mathbf{F} to maximize the expected number of satisfied clauses if we randomly choose z_{i+2}, \dots, z_n .

(We can calculate this efficiently using linearity of expectation.)

The key idea for proving correctness: We fix one variable in each step and keep the expected number of clauses satisfied $\geq 7m/8$.

https://en.wikipedia.org/wiki/Method_of_conditional_probabilities