Sure! Let's break it down with a **bigger example** and a **step-by-step explanation** of **Stochastic Universal Sampling (SUS).**

**Step 1: Understanding the Goal**

SUS is a selection method used in **Genetic Algorithms**. It helps choose individuals **proportionally** to their fitness but ensures that even weaker individuals **have a chance** to be selected.

Imagine you have **six individuals** in a population, each with a different fitness score. Your goal is to select **three** individuals in a way that those with higher fitness get picked more often.

**Step 2: Example Population & Fitness**

**Let's assume we have these six individuals:**

| **Individual** | **Fitness** |
| --- | --- |
| A | 10 |
| B | 20 |
| C | 30 |
| D | 15 |
| E | 5 |
| F | 20 |

🔹 The sum of all fitness values = **10 + 20 + 30 + 15 + 5 + 20 = 100**  
🔹 We want to **select 3 individuals**.

**Step 3: Create a CDF (Cumulative Distribution Function)**

We now transform the fitness scores into a **cumulative fitness array (CDF):**

| **Individual** | **Fitness** | **CDF (Cumulative Sum)** |
| --- | --- | --- |
| A | 10 | 10 |
| B | 20 | 30 |
| C | 30 | 60 |
| D | 15 | 75 |
| E | 5 | 80 |
| F | 20 | 100 |

**Step 4: Generate Evenly Spaced Selection Points**

To pick **3 individuals**, we:

1. **Compute the step size** → **total fitness / num\_selected**

Step size=100/3=33.33\text{Step size} = 100 / 3 = 33.33

1. **Pick a random starting point** between **0 and step size**  
   Suppose we randomly pick **7** as the starting point.
2. **Create three evenly spaced pointers:**
   * **Pointer 1** → **7**
   * **Pointer 2** → **7 + 33.33 = 40.33**
   * **Pointer 3** → **40.33 + 33.33 = 73.67**

**Step 5: Find the Corresponding Individuals**

Now, we **match** each pointer to an individual by finding the first cumulative fitness value that is **greater than or equal** to the pointer.

| **Pointer** | **First CDF ≥ Pointer** | **Selected Individual** |
| --- | --- | --- |
| **7** | **10** | **A** |
| **40.33** | **60** | **C** |
| **73.67** | **75** | **D** |

✅ **Selected Individuals: A, C, D**

**Step 6: Why This Works?**

* The **higher the fitness**, the **higher the chance** of getting selected.
* **No individual is selected more than necessary**, but they get picked **fairly** based on their fitness.
* Unlike **Roulette Wheel Selection**, **SUS ensures a more even spread** by selecting individuals at **regular intervals** in the CDF.

**Python Code for This Example**

import random

import numpy as np

def stochastic\_universal\_sampling(population, fitnesses, num\_selected):

# Convert fitnesses to CDF

cumulative\_fitness = np.cumsum(fitnesses)

total\_fitness = cumulative\_fitness[-1]

# Step size

step\_size = total\_fitness / num\_selected

# Random start point

start\_point = random.uniform(0, step\_size)

pointers = [start\_point + i \* step\_size for i in range(num\_selected)]

# Selection process

selected = []

index = 0

for pointer in pointers:

while cumulative\_fitness[index] < pointer:

index += 1

selected.append(population[index])

return selected

# Example Usage

population = ['A', 'B', 'C', 'D', 'E', 'F']

fitnesses = [10, 20, 30, 15, 5, 20]

num\_selected = 3

selected\_individuals = stochastic\_universal\_sampling(population, fitnesses, num\_selected)

print("Selected Individuals:", selected\_individuals)

**Key Takeaways**

1. **Fitness is converted into a cumulative distribution function (CDF).**
2. **Pointers are spaced evenly**, ensuring fair selection.
3. **Better fitness means higher selection probability**, but weaker individuals still have a chance.
4. **Efficient O(n)O(n) runtime complexity**, better than repeated random sampling.

Would you like a visualization of this process? 🚀

**Understanding the Problem with Fitness-Proportionate Selection (Roulette Wheel & SUS)**

**🚨 Issue: Fitness Scaling Problem**

* In **Roulette Wheel Selection** and **Stochastic Universal Sampling (SUS)**, selection is based on raw fitness values.
* If fitness values are **too close to each other** (e.g., 9.97, 9.98, 9.99), their selection probabilities become almost equal.
* The algorithm starts acting **randomly** instead of favoring the best.
* This problem gets worse as the population **converges**.

**🔧 Fix 1: Fitness Scaling (Not Always a Good Idea)**

* You could **rescale** fitness values to emphasize differences.
* But this still relies on fitness values and doesn’t **fix the core issue**.

**The Real Fix: Rank-Based Selection**

To remove reliance on raw fitness values, we **only care about ranks** (who is better, not how much better).

**💡 Rank-Based Selection (Truncation & Tournament Selection)**

| **Method** | **How It Works** | **Does it rely on raw fitness?** | **Tunable?** |
| --- | --- | --- | --- |
| **Truncation Selection** | Pick the top X% individuals | ❌ No | ⚠️ Only if you change X% |
| **Tournament Selection** | Randomly pick t individuals and return the best | ❌ No | ✅ Yes (change t) |

**🛠️ Tournament Selection: The Best Fix**

Tournament Selection is **the most popular choice** in modern Genetic Algorithms.  
**Why?** ✔ **Ignores actual fitness values**—only compares ranks.  
✔ **No preprocessing**—no need to normalize fitness.  
✔ **Easily tunable**—change t to control selection pressure.  
✔ **Works well in parallel**—fast computation.

**🔹 How Tournament Selection Works**

1️⃣ Randomly **pick t individuals** from the population (with replacement).  
2️⃣ Return **the best** individual among them.

**🔹 Tuning Tournament Selection**

* t = 1 → **Completely random selection** (useless).
* t = 2 → **Balanced selection** (default for GAs).
* t = 7 → **More selective** (used in Genetic Programming).
* **Large t → Behaves like Truncation Selection** (always picks the best).

**🆚 Comparing Selection Methods**

| **Selection Method** | **Uses Fitness Values?** | **Works Well Near Convergence?** | **Computational Cost** | **Best for…** |
| --- | --- | --- | --- | --- |
| **Roulette Wheel Selection** | ✅ Yes | ❌ No (randomizes when fitness is close) | 🔸 Medium | General GA (but not recommended) |
| **Stochastic Universal Sampling (SUS)** | ✅ Yes | ❌ No (same issue as roulette) | 🔸 Medium | Multiple selections at once |
| **Truncation Selection** | ❌ No (only picks top X%) | ✔️ Yes | 🔹 Low | Evolution Strategies |
| **Tournament Selection** | ❌ No (only compares t individuals) | ✔️ Yes | 🔹 Low | General GA (most recommended) |

**🎯 Summary**

🚀 **If you want a robust GA selection method, use Tournament Selection (t=2).**

* It avoids fitness scaling problems.
* It is fast and tunable.
* It doesn’t require fitness normalization.
* It works well in real-world problems.

Would you like a Python implementation of Tournament Selection for your GA? 😊

So always use the portion or truncation selection. Select top k element based on fitness.

**#Research idea we can use SUS or roulette selection. at the beginning since it is fast. Log(n)**

**But by the end we use truncation or torment select as it has precision o(n)**