BANK LENDING DECISIONS OPTIMIZATION

O H M P R O J E C T
I M 3 9 0 0 3

- Penukonda Yeshwanth
19IM30015

Department of Industrial and Systems Engineering

Contents

Introduction

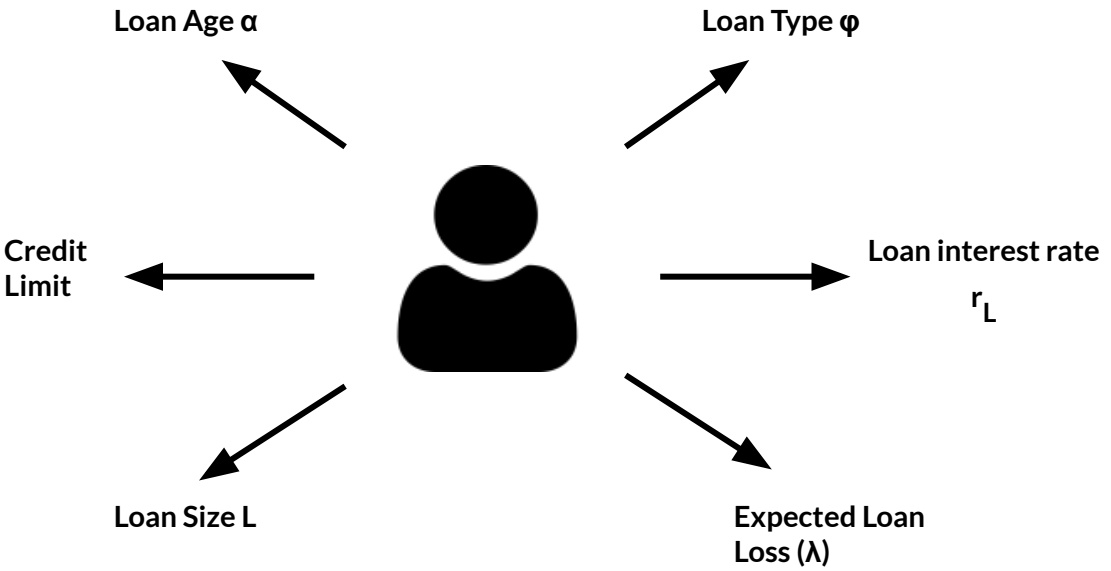
The problem of distributing the limited credit in a way that maximizes profit is very prominent in the banking sector.

A credit crunch is often caused by a sustained period of careless and inappropriate lending, resulting in losses for lending institutions and investors in debt when the loans turn sour and the full extent of bad debts becomes known. Such a situation is highly undesirable and therefore there is a need to set an optimal mechanism of bank lending decisions that will maximize the bank profit in a timely manner.

The problem of bank lending decision in a credit crunch environment- where all applicable customers are eligible to get the desired loan - is an **NP-hard optimization problem** that can be solved using meta-heuristic algorithms. This project explores a few such algorithms to solve the problem.

Problem Description

In this project, the lending decisions are primarily based on the following six characteristics of a customer.



Loan age α

With an assumption that the maximum number of years for any loan is 20 years, the loan age is divided into four categories depending on its expected period time.

Loan age categories.

Category (α)	Value
1	$1 \leq \alpha \leq 3$
2	$3 < \alpha \leq 5$
3	$5 < \alpha \leq 10$
4	$10 < \alpha \leq 20$

Credit limit

The credit limit represents the maximum loan amount that can be given to the customer based on his income and occupation.

Category (L)	Value
Micro	$\$ 0 \leq L \leq \$ 13{,}000$
Small	$\$ 13{,}001 < \alpha \leq \$ 50{,}000$
Medium	$\$ 50{,}001 < \alpha \leq \$ 100{,}000$
Large	$\$ 100{,}001 < \alpha \leq \$ 250{,}000$

Loan size L

Depending on the credit limit, the loan size is determined. The loan size (L) determines the amount of loan requested by a specific customer (C).

Loan type ϕ

Three types of loans are considered : Mortgage (M), Personal (P), and Auto (A).

Loan interest rate rL

Based on the values of ϕ and α, the loan interest rate rL is assigned.

φ	α	r_L Value
M	1	NA
	2	NA
	3	NA
	4	$0.021 \leq r_L \leq 0.028$
P	1	$0.0599 \leq r_L \leq 0.0601$
	2	$0.0601 < r_L \leq 0.0604$
	3	$0.0604 < r_L \leq 0.0609$
A	1	$0.0339 \leq r_L \leq 0.03349$
	2	$0.0349 < r_L \leq 0.0379$
	3	$0.0379 < r_L \leq 0.0399$

Expected loan loss (λ) and borrow credit rating

Borrow Credit Rating is used to measure the range of the expected loan loss (λ).

Credit Rating	λ Value
AAA	$0.0002 \leq \lambda \leq 0.0003$
AA	$0.0003 < \lambda \leq 0.001$
A	$0.001 < \lambda \leq 0.0024$
BBB	$0.0024 < \lambda \leq 0.0058$
BB	$0.0058 < \lambda \leq 0.0119$

Dataset Provided

Customer No	Interest	Rating	Loss (λ)	Loan Size
1	0.021	AAA	0.0002	10
2	0.022	BB	0.0058	25
3	0.021	A	0.0001	4
4	0.027	AA	0.0003	11
5	0.025	BBB	0.0024	18
6	0.026	AAA	0.0002	3
7	0.023	BB	0.0058	17
8	0.021	AAA	0.0002	15
9	0.028	A	0.001	9
10	0.022	A	0.001	10

Fitness Function

The fitness function employed to compare two solutions in all the algorithms used in this project, is made of the following components :

- **Loan Revenue**

$$\vartheta = \sum_{i=0}^{n} (r_L L - \lambda)$$

- **Loan Cost**

$$\mu = \sum_{i=0}^{n} L\delta$$

- **Total transaction cost**

$$\varpi = \sum_{i=0}^{n} r_T T$$

- **Cost of demand deposit**

$$\beta = r_D D$$

$$F = \vartheta + \mu + \varpi - \beta - \sum_{i=0}^{n} \lambda$$

D	financial institution's deposit
K	required reserve ratio
r_D	return on deposit
δ	predetermined institutional cost
r_T	customer transaction rate

Formulation

For, **x** → $[x_1, x_2, x_3, \ldots\ldots x_n]$; n → number of customers

Maximize $\sum(r_{L_i}L_i - \lambda_i)x_i + \sum\delta L_i x_i + \sum r_T((1-k)D - L_i)x_i - r_D D - \sum x_i\lambda_i$

Where

L_i = Loan size of customer 'i'

λ_i = Loss corresponding to customer 'i'

$X_i \in [0,1]$ → binary variable

Subject to $(1-k)D \geq \sum L_i x_i$

* Constraint handling mechanism described in the Annexure

Solution X is represented as a binary encoded vector of size 'n' where n corresponds to the number of customers.

For examples, a possible solution for a customer pool of size 10 can be represented as ,

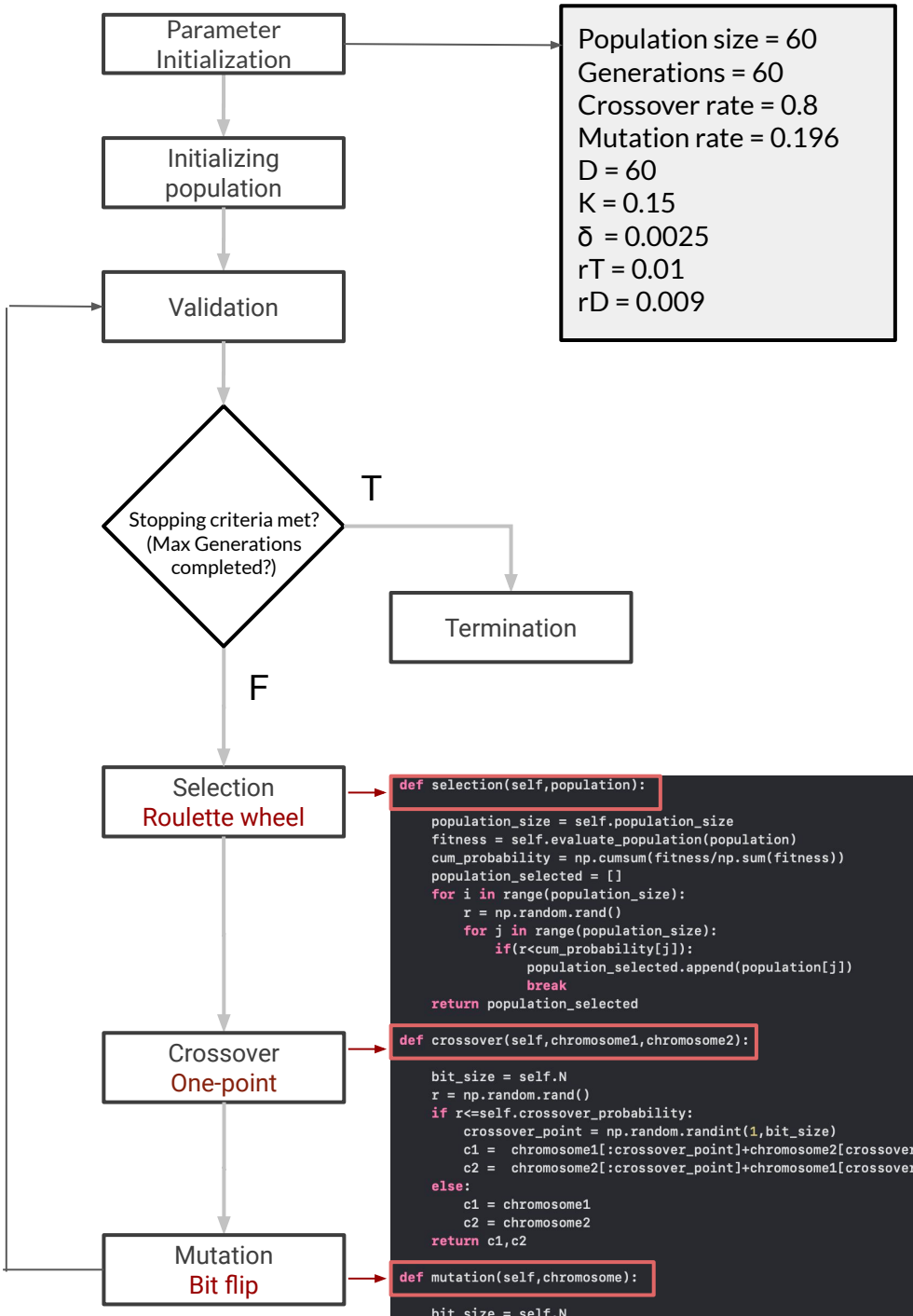
0	1	0	1	1	0	1	1	0	1

Indicating that customers ***2,4,5,7,8 & 10 are selected***

Approach 1 : Genetic Algorithm

class GA

The process of implementing GA is described in the flowchart below :




```python
def selection(self,population):

    population_size = self.population_size
    fitness = self.evaluate_population(population)
    cum_probability = np.cumsum(fitness/np.sum(fitness))
    population_selected = []
    for i in range(population_size):
        r = np.random.rand()
        for j in range(population_size):
            if(r<cum_probability[j]):
                population_selected.append(population[j])
                break
    return population_selected
```

```python
def crossover(self,chromosome1,chromosome2):

    bit_size = self.N
    r = np.random.rand()
    if r<=self.crossover_probability:
        crossover_point = np.random.randint(1,bit_size)
        c1 =  chromosome1[:crossover_point]+chromosome2[crossover_point:]
        c2 =  chromosome2[:crossover_point]+chromosome1[crossover_point:]
    else:
        c1 = chromosome1
        c2 = chromosome2
    return c1,c2
```

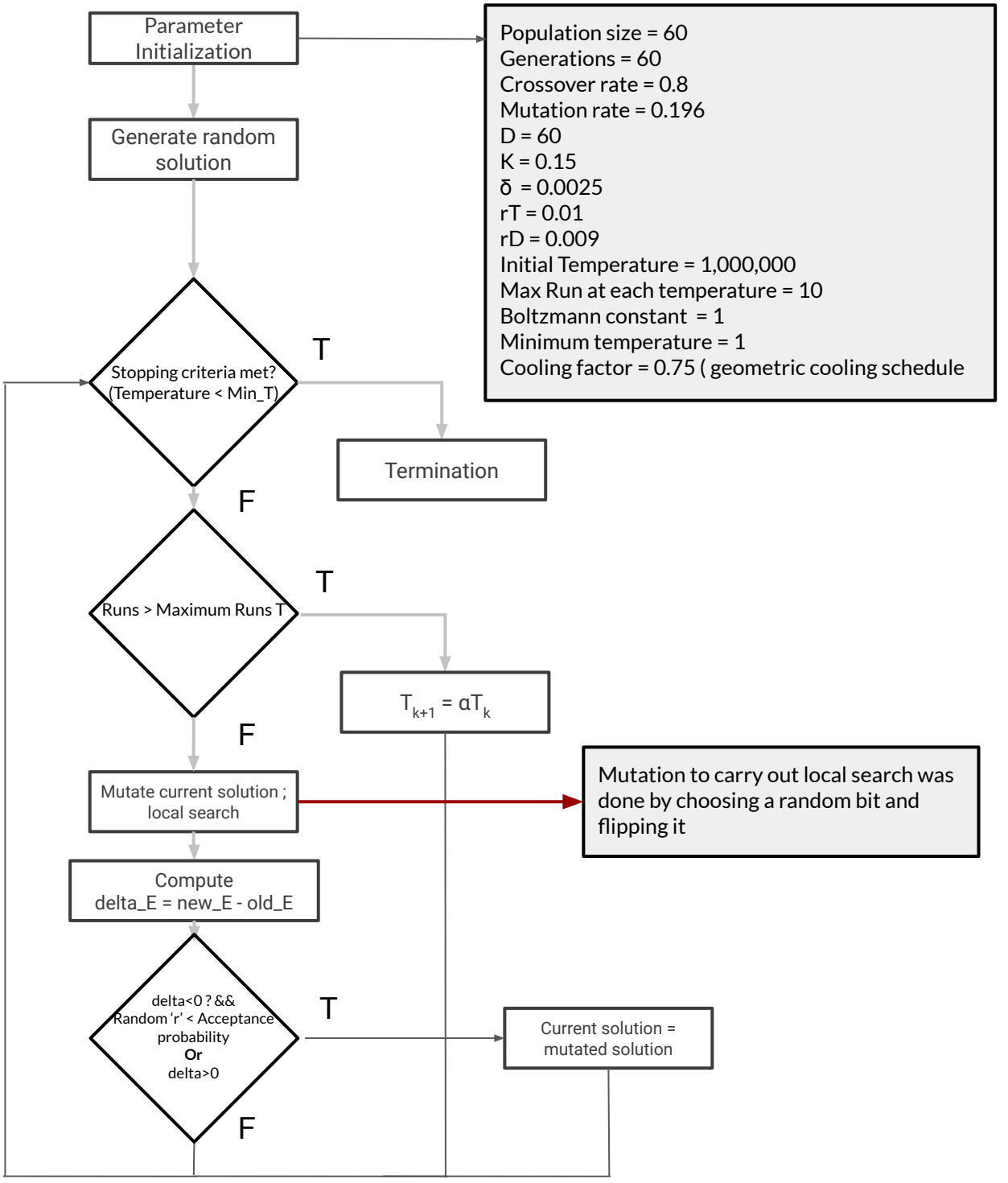
```python
def mutation(self,chromosome):

    bit_size = self.N
    r = np.random.rand()
    if r<=self.mutation_probability:
        mutation_point = np.random.randint(bit_size)
        chromosome[mutation_point] =  1-chromosome[mutation_point]
    return chromosome
```

Approach 2 : Simulated Annealing

```
class SA
```

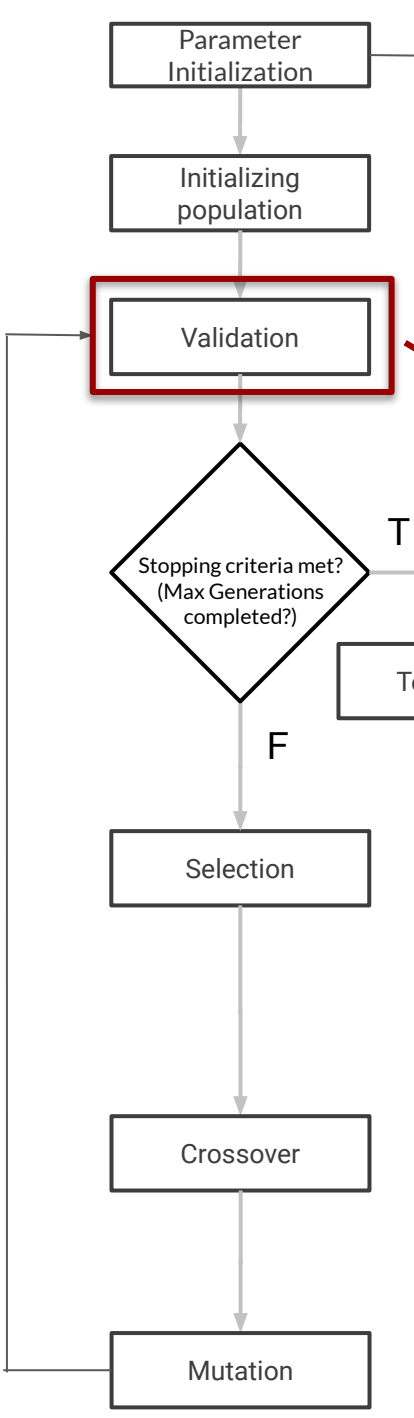
The process of implementing Simulated Annealing is described in the flowchart below :



Approach 3 : Genetic Simulated Annealing

`class Gene_SA`

The process of implementing Genetic Simulated Annealing is described in the flowchart below :



Population size = 60
Generations = 60
Crossover rate = 0.8
Mutation rate = 0.196
D = 60
K = 0.15
δ = 0.0025
rT = 0.01
rD = 0.009

Initial Temperature = 1,000
MaxRun at a temperature = 20
Boltzmann constant = 1
Minimum temperature = 1
Cooling factor = 0.3 (geometric cooling schedule

What's different in GeneSA?

The validation process

While a conventional Genetic Algorithm strives to improve the quality of its population with each generation, Simulated annealing accepts a not so good solution in hopes of landing a better solution.

Combining both the above concepts, we validate the population of each generation by comparing the median of the population to that of previous generation.

Following are the steps of validation :

1. Compute
   delta_E = median fitness(new_population)-median fitness(old_population)
2. If delta_E>0 → proceed with the new population
3. If delta_E<0 → proceed with new population with a probability $P_A = e^{delta\_E/T}$
4. Else : continue with the old population

Why compare median fitness of populations to decide acceptance of a population?

Ensuring that at least 50% of the chromosomes in the new population have a better fitness than 50% of the old population guarantees that the quality of the population improves constantly.

Using the simulated annealing approach, we may accept a population with lower median fitness in hopes of getting a much better set of chromosomes in the subsequent generations.

Results

```
The best fitness obtained  3.3028999999999997
The best portfolio obtained  [1, 0, 1, 1, 0, 1, 0, 0, 1, 1]
```

The best fitness value obtained was : 3.302899

Best Allocation found :

1	0	1	1	0	1	0	0	1	1

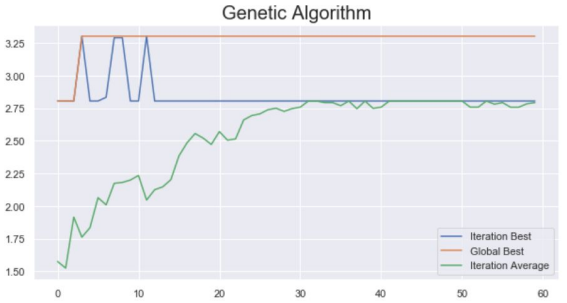
Customers 1,3,4,6,9 & 10 are to be selected to maximize fitness

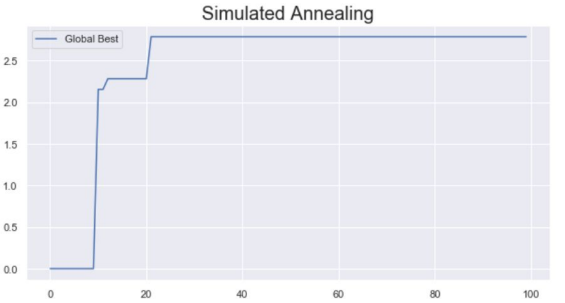
It is also interesting to note that only customers with AA/AAA/A credit rating were selected even though B/BB/BBB rating could bring in higher revenue owing to higher loan sizes

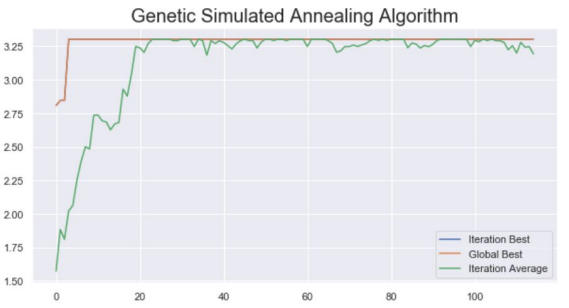
Visualization of results & Comparing Algorithms

The global best fitness, iteration fitness and average fitness of the population has been plotted against each generation for GA & GeneSA algorithms.

Global best for SA has been plotted against iterations



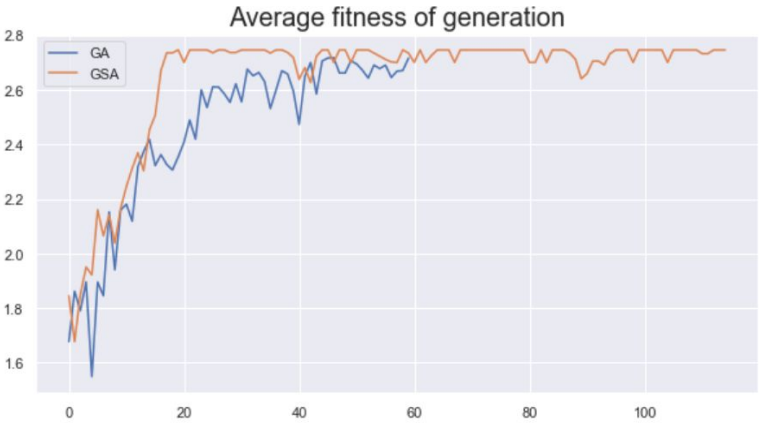


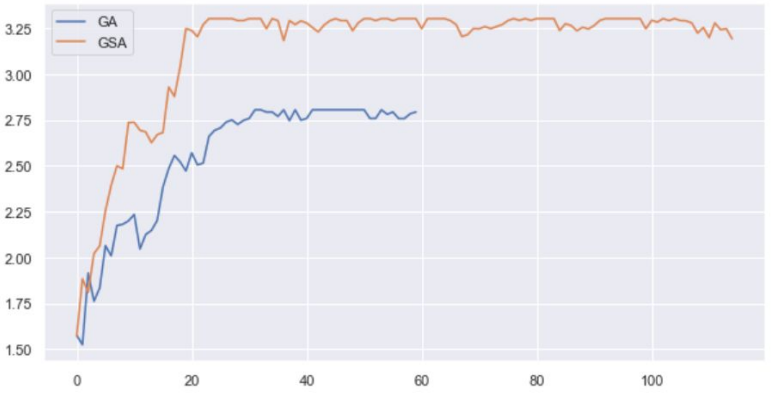


It is seen from the above figures that both GA and GeneSA reach the global optimal within much lesser iterations as compared to SA.

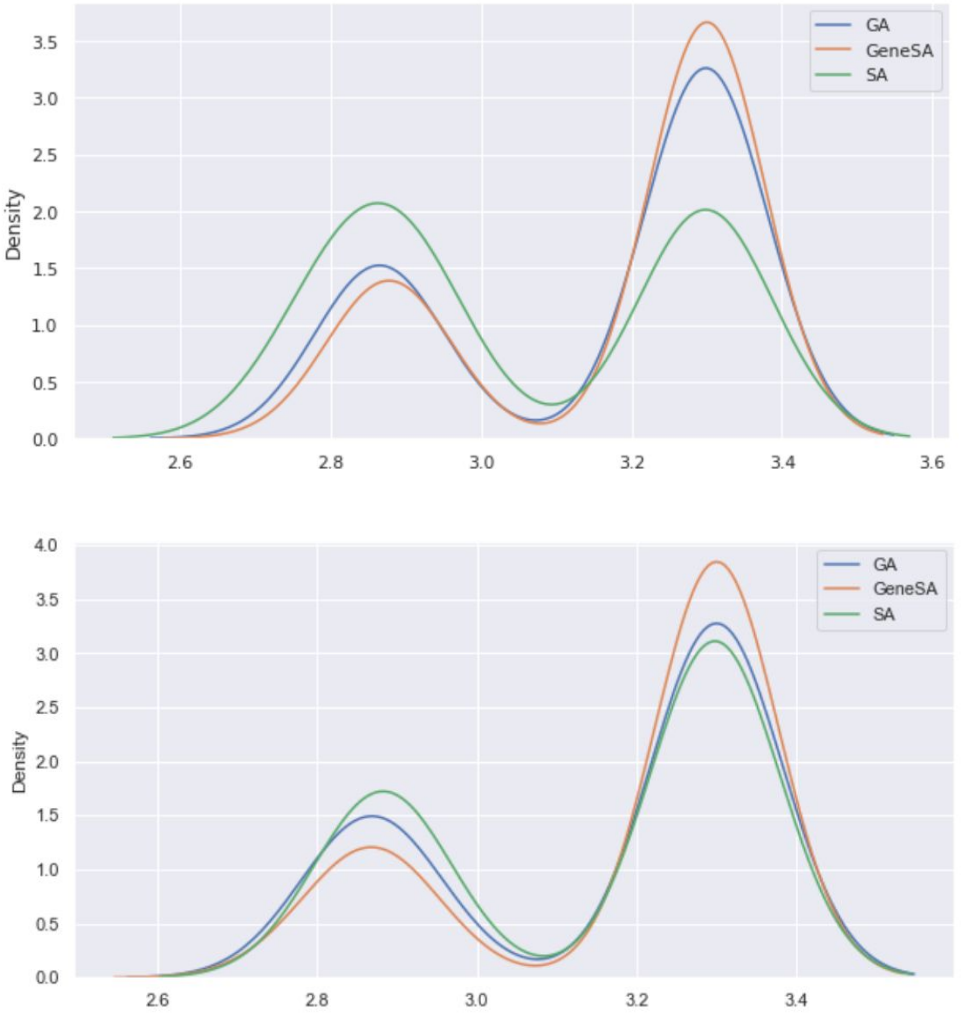
This can be attributed to the fact that each iteration of the former checks a total of 60 solutions (=population size), while the conventional SA algorithms checks only 1 solution per iteration.

Additionally when we compare the average fitness of GA and GeneSA, we see that GeneSA converges to the optimal value while the overall quality of the final generation remains suboptimal for GA. This essentially implies that **GeneSA improves the exploitation capability of the optimization algorithm. By being selective of the population to retain only the best chromosomes, GeneSA produced high quality individuals by the time of termination.**





Analyzing just one complete run of **probabilistic algorithms** such Genetic Algorithm and Simulated Annealing will not let us know the full story. So to understand how efficient each of the three algorithms were in yielding the optimal solution, **each algorithm was run for a total of 100 times** and the best fitness obtained for each run was recorded.



As we can see from both the figures above, in a total of 100 runs of each algorithm, GeneSA was able to reach optimal the highest number of times, followed by GA. Simulated annealing showed the least probability of reaching the optimal, and instead showed a higher probability of reaching the nearest local optimal.

Conclusion :

- ➢ The probability of reaching the global optimal was in the order → GeneSA > GA > SA
- ➢ SA showed the slowest convergence rate
- ➢ Alongwith convergence to the global optimal, GeneSA yielded a much better quality of individuals in the end as compared to GA

Annexure

Two ways of constraint handling were tried with all the 3 algorithms

1. **Penalty Function added to the fitness function**

   Penalty function P(x) = **10* min (** (1-k)D-L, 0**)**

2. **Checking feasibility of every solution considered**

   Defined a feasibility function that checks for validity of every chromosome generated after crossover and mutation, and the infeasible ones were discarded and replaced by redoing the process

Out of the methods listed above, penalty function did a better job at converging to the global optimum. This is because penalty function facilitates more exploration by keeping the infeasible ones in the population. Checking for feasibility yielded sub-optimal solutions in majority of the runs of all 3 algorithms since exploration is highly inhibited. Additionally this step is highly computationally expensive.

Therefore, penalty functions are a much better way of handling constraints as compared to feasibility check.