```
# IMPORTANT: SOME KAGGLE DATA SOURCES ARE PRIVATE
# RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES.
import kagglehub
kagglehub.login()
```

```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

harshanrs_matrix_path = kagglehub.dataset_download('harshanrs/matrix')

print('Data source import complete.')
```

```
#CS547 – ADVANCED TOPICS IN SOFTWARE ENGINEERING
#ASSIGNMENT 1 - Part 2
#Group: CS547Assignment1Part2Group47
#Members:
#1. HARSHAN RETHINAVELU SELVAKUMAR – 202480548
#2. MANOJ KUMAR DHARMARAJ - 202468855
```

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Sa
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/kaggle/input/matrix/bigfaultmatrixplustime.txt
/kaggle/input/matrix/smallfaultmatrixplustime.txt
```

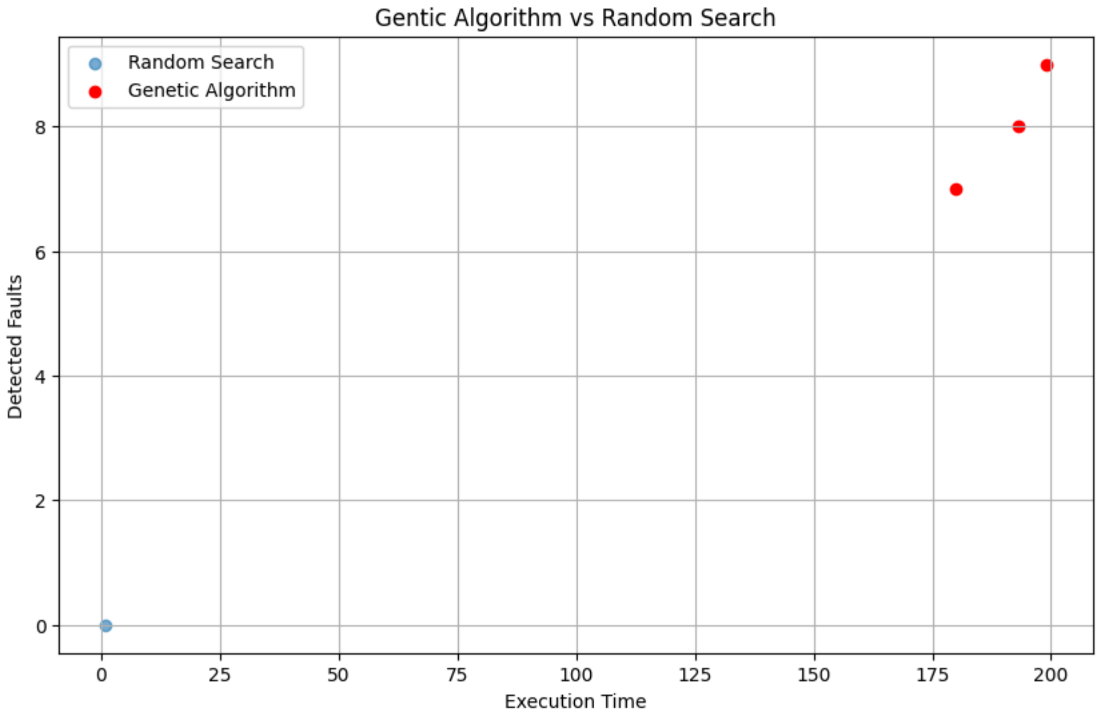
```
#Importing the necessary libraries
import random
import matplotlib.pyplot as plt
from deap import creator,base,tools,algorithms

#Function that loads the data of the file including the execution time
def load_file(filepath):

    #Create an empty list
    tests=[]

    with open(filepath,'r') as file:
        for line in file:

            #Taking each line of the file seperately, removing any leading whitespaces
            #Splitting the line into list of strings using commas
            seperate=line.strip().split(',')
            test_num=seperate[0]

            #Faults upto the last element
            fault=list(map(int,seperate[1:-1]))

            #Execution time(last element)
            execution_time=float(seperate[-1])

            #Adding the test id, fault and execution time
            tests.append((test_num,fault,execution_time))

    return tests

#Fitness function to maximize the number of faults detected and minimize the total execution time
```

```
def fit(individual,tests):

    #Creating an empty list to store the selected tests
    sel_tests=[]

    for i in range(len(individual)):
        if individual[i]==1:
            sel_tests.append(tests[i])

    tot_execution_time=0

    #Tracking unique faults detected
    detected_faults=set()

    #Looping through each selected test
    for _,faults,execution_time in sel_tests:
        tot_execution_time+=execution_time

        #Update the list (Unique faults)
        for index,fault in enumerate(faults):
            if fault==1:
                detected_faults.add(index)

    #Total number of faults detected
    tot_detected_faults=len(detected_faults)

    #Return the objectives
    return tot_detected_faults,tot_execution_time



#Creating DEAP classes
creator.create("FitnessMulti",base.Fitness,weights=(1.0,-1.0))
creator.create("Individual",list,fitness=creator.FitnessMulti)


#Defining DEAP toolbox
def tbox(tests):
    toolbox=base.Toolbox()

    toolbox.register("attr_bool",random.randint,0,1)
    toolbox.register("individual",tools.initRepeat,creator.Individual,toolbox.attr_bool,n=len(tests))
    toolbox.register("population",tools.initRepeat,list,toolbox.individual)
    toolbox.register("evaluate",fit,tests=tests)

    #Defining the genetic operators
    toolbox.register("mate",tools.cxUniform,indpb=0.5)
    toolbox.register("mutate",tools.mutFlipBit,indpb=0.2)
    toolbox.register("select",tools.selNSGA2)
    return toolbox


#Genetic Algorithm
def ga(toolbox,tests,pop_size=50,gen=25,cxpb=0.7,mutpb=0.3):

    #Initializing the population
    pop=toolbox.population(n=pop_size)

    #Evaluating the initial population
    for ind in pop:
        ind.fitness.values=toolbox.evaluate(ind)


    #Running the GA
    algorithms.eaMuPlusLambda(pop,toolbox,mu=pop_size,lambda_=pop_size,
                              cxpb=cxpb,mutpb=mutpb,ngen=gen,verbose=False)


    #Pareto Front
    pareto=tools.sortNondominated(pop,len(pop),first_front_only=True)[0]

    print(f"\nResults for GA (pop_size={pop_size},gen={gen},cxpb={cxpb},mutpb={mutpb}):")
    print(f"\nTotal final solutions from pareto front:{len(pareto)}")

    #Printing the best individuals
    print("Best Individuals in pareto front:")

    for i,individual in enumerate(pareto,start=1):
        sel_tests=[tests[index][0] for index,bit in enumerate(individual) if bit==1]
        print(f"\tIndividual{i}:{sel_tests}")
        print(f"\tIndividual{i} Fitness:{individual.fitness.values}\n")

    return pareto
```

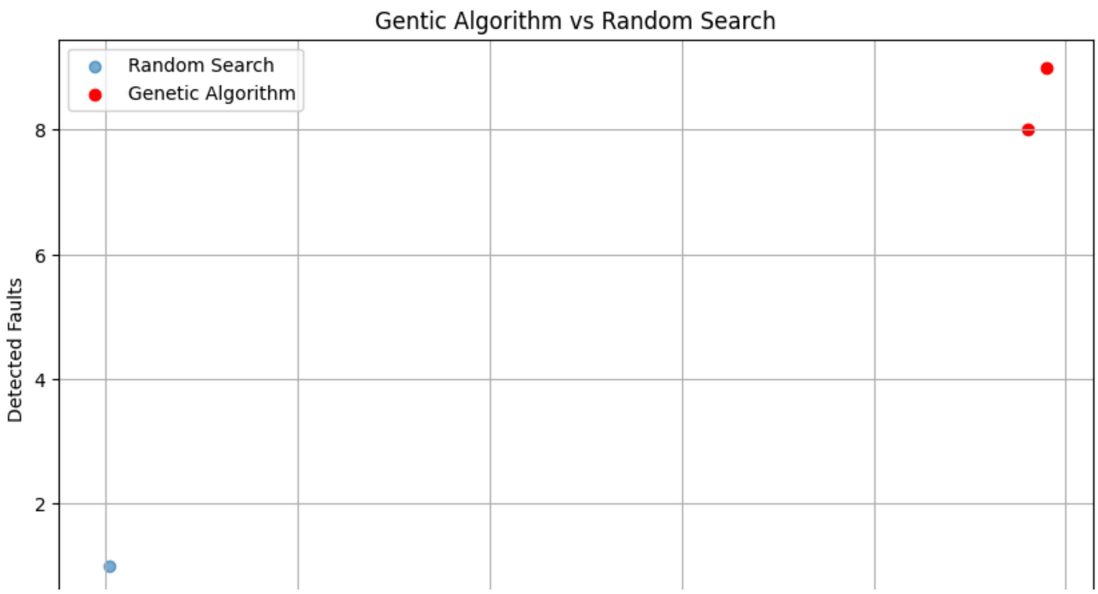
```
#Function to compute the metrics from the pareto front
def cal_metrics(pareto):

    #Average of faults detected
    faults_avg=np.mean([ind.fitness.values[0] for ind in pareto])

    #Average execution time
    time_avg=np.mean([ind.fitness.values[1] for ind in pareto])

    #Max of faults detected
    faults_max=max([ind.fitness.values[0] for ind in pareto])

    #Minimum execution time
    time_min=min([ind.fitness.values[1] for ind in pareto])

    return{"Average of faults detected": faults_avg,
          "Average execution time": time_avg,
          "Max of faults detected": faults_max,
          "Minimum Execution time": time_min}


#Random Search
from deap.creator import Individual
def random_search(tests,runs=10):

    #Creating a list to store the results
    result=[]
    for _ in range(runs):

        #Random Individual
        individual=[random.randint(0,1) for _ in range(len(tests))]

        #Evaluating the fitness
        faults,time=fit(individual,tests)

        #Creating DEAP individual
        ind=Individual(individual)

        #Fitness
        ind.fitness.values=(faults,time)
        result.append(ind)

    #Pareto front
    pareto=tools.sortNondominated([sol for sol in result],len(result),first_front_only=True)[0]

    return pareto,result


#Visualization
def visualize(ga_pareto,random_pareto):
    fault_GA=[ind.fitness.values[0] for ind in ga_pareto]
    time_GA=[ind.fitness.values[1] for ind in ga_pareto]

    fault_random=[sol[0] for sol in random_pareto]
    time_random=[sol[1] for sol in random_pareto]

    plt.figure(figsize=(10,6))
    plt.scatter(time_random,fault_random,label="Random Search",alpha=0.6)
    plt.scatter(time_GA,fault_GA,label="Genetic Algorithm",marker='o',color='red')
    plt.xlabel("Execution Time")
    plt.ylabel("Detected Faults")
    plt.title("Gentic Algorithm vs Random Search")
    plt.legend()
    plt.grid()
    plt.show()


#Main funtion
if __name__=="__main__":

    #Reading the datasets
    test1=load_file("/kaggle/input/matrix/smallfaultmatrixplustime.txt")
    test2=load_file("/kaggle/input/matrix/bigfaultmatrixplustime.txt")

    #Parameters sets for diverse solutions
    parameters=[{"pop_size":50,"gen":25,"cxpb":0.7,"mutpb":0.3},
               {"pop_size":50,"gen":25,"cxpb":0.6,"mutpb":0.4},
               {"pop_size":50,"gen":25,"cxpb":0.8,"mutpb":0.2},]
```

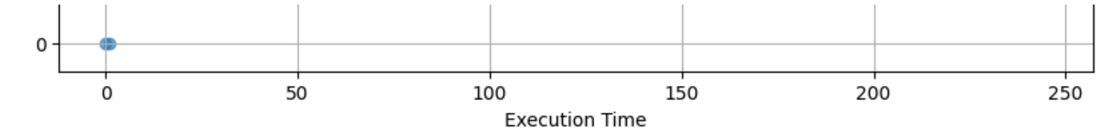
```python
#Using for loop to go through the datasets and parameters
for dataset,tests in [("smallfaultmatrixplustime",test1),("bigfaultmatrixplustime",test2)]:
    print(f"\n{dataset}:")

    #Creating toolbox
    toolbox=tbox(tests)

    for parameter in parameters:

        #Genetic Algorithm...
        pareto=ga(toolbox,tests,pop_size=parameter["pop_size"],
                  gen=parameter["gen"],cxpb=parameter["cxpb"],mutpb=parameter["mutpb"])

        #Computing metrics for GA
        ga_metrics=cal_metrics(pareto)
        print(f"\nGA Metrics:" )
        print(ga_metrics)

        #Random Search...
        pareto_random,result=random_search(tests,runs=10)

        #Visualizing the results
        visualize(pareto,pareto_random)
```

```
smallfaultmatrixplustime:

Results for GA (pop_size=50,gen=25,cxpb=0.7,mutpb=0.3):

Total final solutions from pareto front:3
Best Individuals in pareto front:
        Individual1:['t0', 't6', 't14', 't19', 't24', 't33', 't36', 't44', 't52', 't56', 't59', 't65', 't67', 't72', 't77', 't88',
        Individual1 Fitness:(8.0, 193.0)

        Individual2:['t0', 't1', 't6', 't14', 't16', 't20', 't24', 't32', 't35', 't36', 't44', 't52', 't59', 't73', 't77', 't88', 't
        Individual2 Fitness:(7.0, 180.0)

        Individual3:['t0', 't1', 't4', 't6', 't14', 't16', 't20', 't24', 't36', 't40', 't44', 't45', 't52', 't55', 't56', 't59', 't6
        Individual3 Fitness:(9.0, 199.0)


GA Metrics:
{'Average of faults detected': 8.0, 'Average execution time': 190.66666666666666, 'Max of faults detected': 9.0, 'Minimum Execution
```

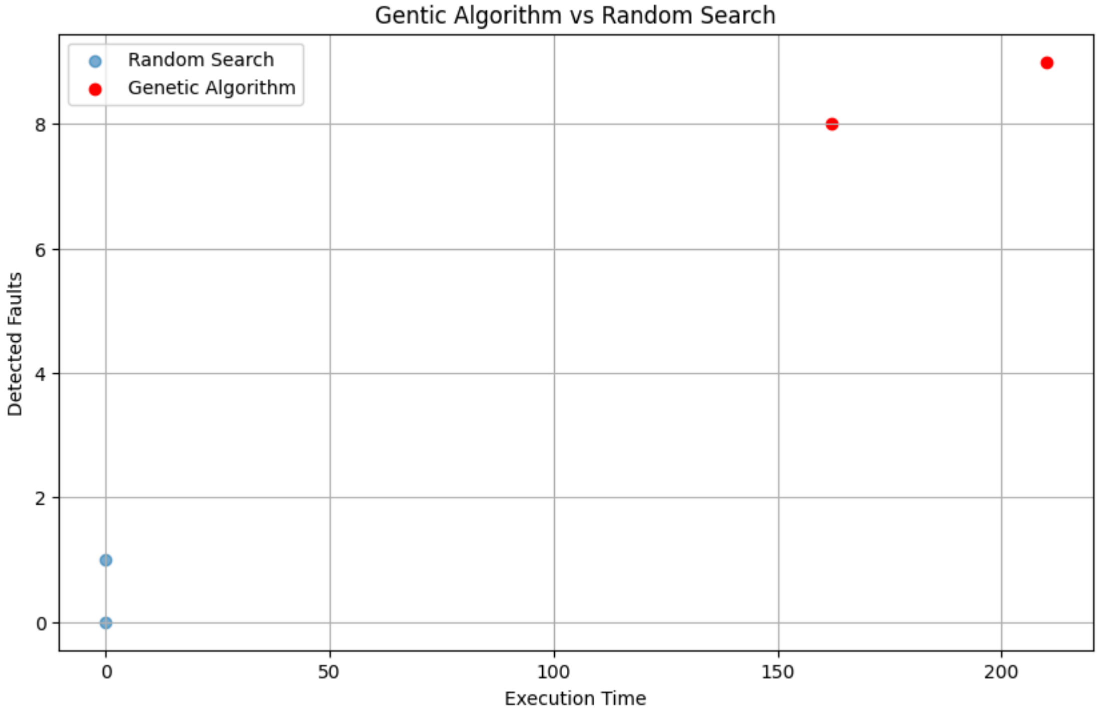



```
Results for GA (pop_size=50,gen=25,cxpb=0.6,mutpb=0.4):

Total final solutions from pareto front:2
Best Individuals in pareto front:
        Individual1:['t0', 't3', 't4', 't11', 't12', 't15', 't16', 't27', 't35', 't36', 't37', 't42', 't43', 't44', 't45', 't48', 't
        Individual1 Fitness:(9.0, 245.0)

        Individual2:['t0', 't1', 't12', 't30', 't35', 't36', 't37', 't42', 't44', 't45', 't49', 't50', 't52', 't54', 't57', 't62',
        Individual2 Fitness:(8.0, 240.0)


GA Metrics:
{'Average of faults detected': 8.5, 'Average execution time': 242.5, 'Max of faults detected': 9.0, 'Minimum Execution time': 240.0}
```

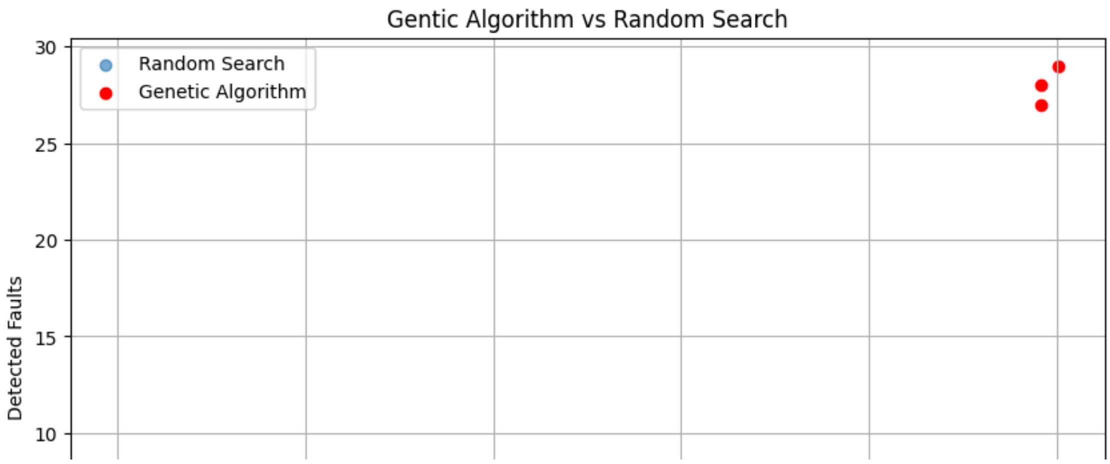


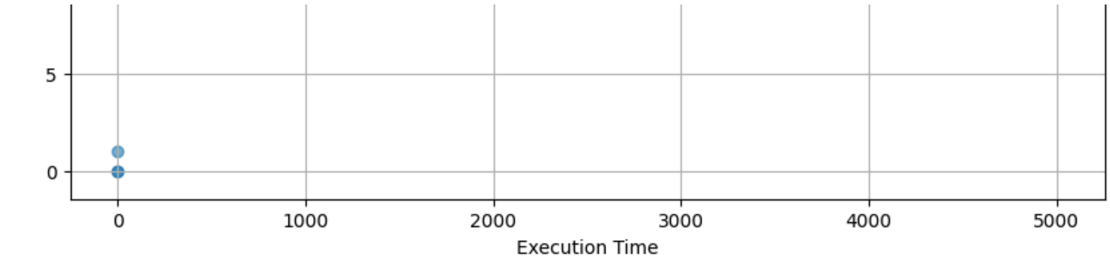



```
Results for GA (pop_size=50,gen=25,cxpb=0.8,mutpb=0.2):

Total final solutions from pareto front:2
Best Individuals in pareto front:
        Individual1:['t1', 't4', 't13', 't15', 't24', 't28', 't31', 't35', 't36', 't41', 't50', 't52', 't54', 't59', 't61', 't69',
        Individual1 Fitness:(8.0, 162.0)

        Individual2:['t3', 't6', 't12', 't13', 't15', 't28', 't31', 't34', 't36', 't48', 't52', 't58', 't59', 't62', 't69', 't78',
        Individual2 Fitness:(9.0, 210.0)


GA Metrics:
{'Average of faults detected': 8.5, 'Average execution time': 186.0, 'Max of faults detected': 9.0, 'Minimum Execution time': 162.0}
```

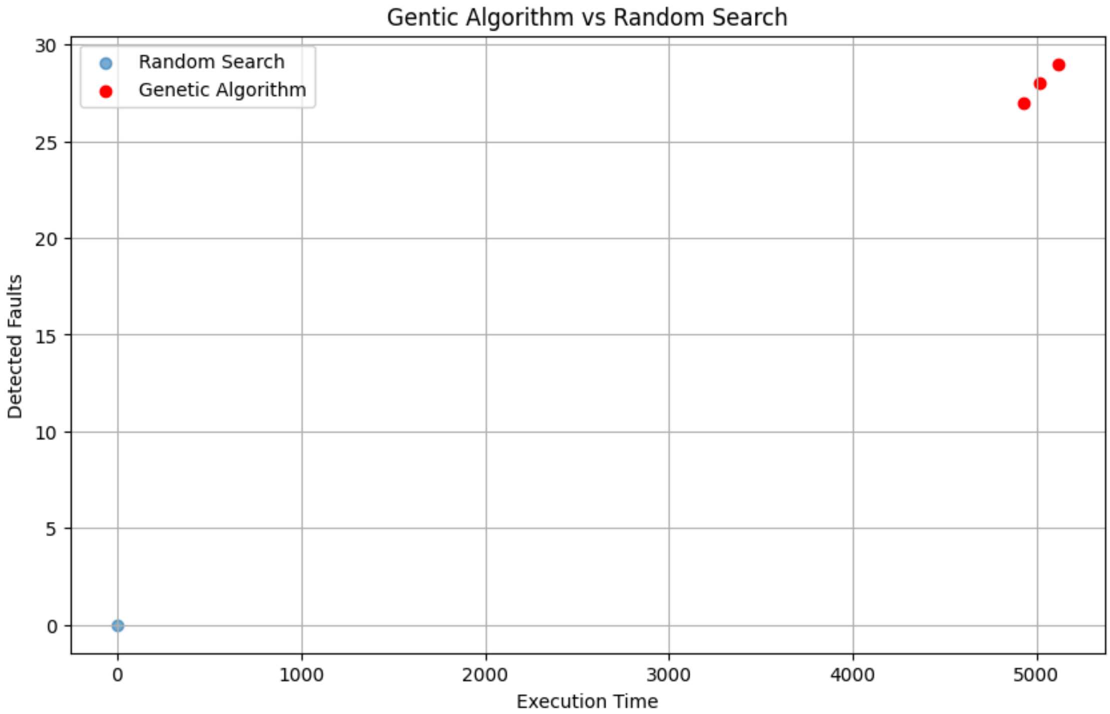



```
bigfaultmatrixplustime:

Results for GA (pop_size=50,gen=25,cxpb=0.7,mutpb=0.3):

Total final solutions from pareto front:3
Best Individuals in pareto front:
        Individual1:['t1', 't10', 't1000', 't10001', 't10007', 't10009', 't10010', 't10013', 't10015', 't10017', 't10020', 't10021',
        Individual1 Fitness:(29.0, 5007.0)

        Individual2:['t1', 't10', 't100', 't1000', 't10001', 't10007', 't10009', 't10010', 't10015', 't10016', 't10017', 't10020',
        Individual2 Fitness:(28.0, 4916.0)

        Individual3:['t100', 't1000', 't10001', 't10002', 't10003', 't10007', 't10009', 't10010', 't10014', 't10015', 't10016', 't1
        Individual3 Fitness:(27.0, 4914.0)


GA Metrics:
{'Average of faults detected': 28.0, 'Average execution time': 4945.666666666667, 'Max of faults detected': 29.0, 'Minimum Executio
```

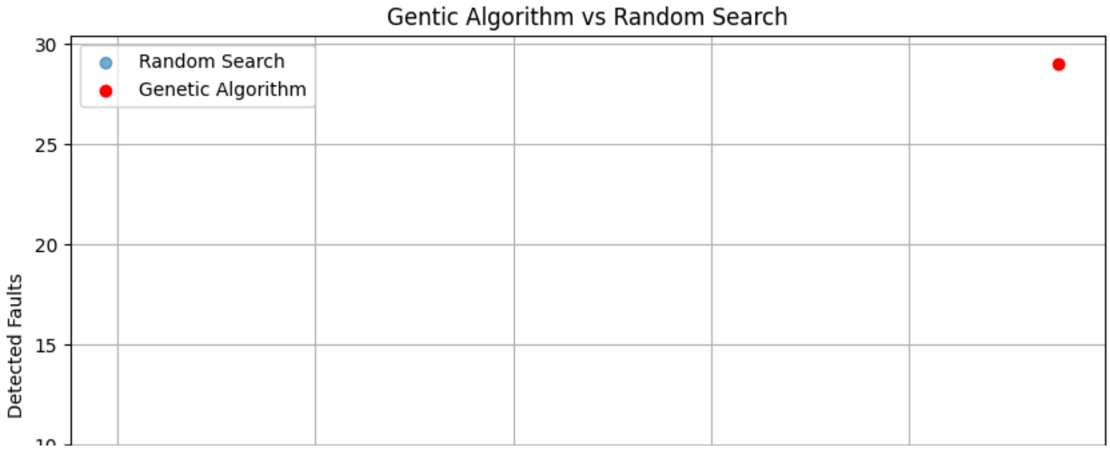


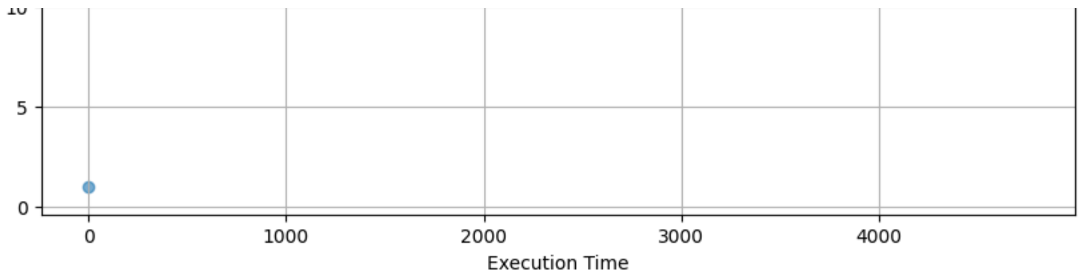



```
Results for GA (pop_size=50,gen=25,cxpb=0.6,mutpb=0.4):

Total final solutions from pareto front:3
Best Individuals in pareto front:
        Individual1:['t100', 't10000', 't10001', 't10004', 't10007', 't10009', 't10012', 't10013', 't10014', 't10015', 't10021', 't1
        Individual1 Fitness:(29.0, 5115.0)

        Individual2:['t10001', 't10002', 't10003', 't10006', 't10007', 't10009', 't1001', 't10011', 't10013', 't10014', 't10016', 't
        Individual2 Fitness:(28.0, 5015.0)

        Individual3:['t10001', 't10002', 't10003', 't10005', 't10006', 't1001', 't10013', 't10014', 't10016', 't1002', 't10020', 't1
        Individual3 Fitness:(27.0, 4927.0)


GA Metrics:
{'Average of faults detected': 28.0, 'Average execution time': 5019.0, 'Max of faults detected': 29.0, 'Minimum Execution time': 492
```




```
Results for GA (pop_size=50,gen=25,cxpb=0.8,mutpb=0.2):

Total final solutions from pareto front:1
Best Individuals in pareto front:
        Individual1:['t1000', 't10003', 't10007', 't10009', 't10011', 't10016', 't10017', 't10022', 't10029', 't10032', 't10036', 't
        Individual1 Fitness:(29.0, 4752.0)


GA Metrics:
{'Average of faults detected': 29.0, 'Average execution time': 4752.0, 'Max of faults detected': 29.0, 'Minimum Execution time': 475
```




10
5
0
0
1000
2000
3000
4000
Execution Time

```
#References
#https://github.com/DEAP/deap
#https://advancedoracademy.medium.com/multi-objective-optimization-a-comprehensive-guide-with-python-example-09edc2af03f3
#http://deap.gel.ulaval.ca/doc/default/api/algo.html
```