Test Case Prioritization using Meta-heuristic Algorithm

A PROJECT REPORT

Submitted by

S.Sharavanan
14mse1140

School of Computing Science and Engineering

Vellore Institute of Technology

Vandalur - Kelambakkam Road, Chennai - 600 127

April - 2019

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computing Science and Engineering

DECLARATION

I hereby declare that the project entitled Test Case Prioritization using Meta-Heuristic Algorithm submitted by me to the School of Computing Science and Engineering, VIT Chennai, 600 127 in partial fulfillment of the requirements of the award of the degree of Master of Science in Software Engineering (5 year Integrated Programme) is a bona-fide record of the work carried out by me under the supervision of Prof. Dr.Poongodi.M. I further declare that the work reported in this project, has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or of any other institute or University.

Place: Chennai
Date:

Signature of Candidate
(Sharavanan.S)

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computing Science and Engineering

CERTIFICATE

This is to certify that the report entitled **Test Case Prioritization using Meta-heuristic Algorithm** is prepared and sub-mitted by **S.Sharavanan** (Reg. No. **14mse1140**) to VIT Chennai, in partial fulfillment of the requirement for the award of the degree of Master of Science in Software Engineering (5 year Integrated Programme) is a bona-fide record carried out under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Guide/Supervisor	Program Chair
Name: Prof.B.Prakash	Name: Dr. V. Viswanathan
Date:	Date:
Examiner	Examiner
Name:	Name:
Date:	Date:

(Seal of SCSE)

Acknowledgement

I obliged to give my appreciation to a number of people without whom I could not have completed this thesis successfully.

I would like to place on record my deep sense of gratitude and thanks to my internal guide **Prof.B.Prakash**, School of Computer Science and Engineering (SCSE), VIT Chennai whose esteemed support and immense guidance encouraged me to complete the project successfully.

I would like to thank our Program chair **Dr.V.Viswanathan**, School of Computer Science and Engineering (SCSE) and Project Coordinator **Dr.Poongodi.M**, VIT Chennai, for their valuable support and encouragement to take up and complete this thesis.

Special mention to our dean to **Dr.Vaidehi Vijayakumar** , Associate Deans **Dr. Vijayakumar V** and **Dr.Subbulakshmi T,** School of Computer Science and Engineering (SCSE), VIT Chennai, for spending their valuable time and efforts in sharing his knowledge and for helping me in every minute aspect of software engineering.

I thank our management of VIT Chennai campus for permitting me to use the library resources. I also thank all the faculty members for giving me the courage and the strength that I needed to complete my goal. This acknowledgment would be incomplete without expressing the whole hearted thanks to my family and friends who motivated me during the course of my work.

S.Sharavanan
14MSE1140

Abstract

In Software Testing, ‘Test case’ is a set of conditions or variable used to determine whether a system under test satisfies requirements and works correctly.

The objective of this work is to generate the set of test cases for the identified domain or project, prioritize the test cases using meta-heuristic technique. Further, the prioritized test cases are evaluated by comparing with other evolutionary algorithms based on running time, time complexity and accuracy.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Software Testing is the process of evaluation the software application's functionality, and to find whether the developed software met the specified requirements and identify the detect and free in order to produce the quality product. The effective software test methods and automation tools are strongly needed in the real world in order to deliver high quality software products. Testing is the process identifies faults and by removing faults we can increase the quality of software, it's also measures the capability for achieving accurateness, reliability, usability, maintainability, reusability, correctness and testability. Test Case Prioritization is process to prioritize the tester's test cases to measure and run the test cases in the regression testing process and also increase a test suite's rate of fault detection. Regression Testing has three processes: Test suite minimization; Test case selection; Test case prioritization.

1.1 Background

In today's updated world, PSO algorithm is a method used to find the search space of the problem to find the settings and given parameters. Normally the test cases are generated to prioritize from some other or domain where they pull up a set of test plans for particular projects or domains. Then, the test cases are generated it's used to prioritize from the fault and test coverage. Each test case has the particular information about project and it's implemented in programming language.

1.2 Problem Statement

The following are the objectives of this study:

1) Generate the set of test cases
2) Prioritize the test cases using PSO algorithm
3) Performance of the proposed algorithm will be evaluated based on running time, time complexity and accuracy.

1.3 Motivation

There are several evolutionary algorithm utilized in the field of software engineering domain. Each algorithm has its own advantage and limitations and no single algorithm is suitable for all type of optimization problem. This motivates us to perform this study to understand the most efficient evolutionary algorithm for prioritizing the test cases in the software development process

1.4 Challenges

One of the major challenge faced as elicitation of test cases dataset for the selected project due to availability of several meta-heuristic algorithm selecting the most appropriate algorithm for optimization is another challenge

1.5 Proposed System

The objective of this study is as follows

- To generate the set of test cases for the identified domain or project
- Prioritize the test cases using meta-heuristic technique.
- The proposed approach is evaluated by comparing with other evolutionary algorithm (Ant colony Optimization-ACO) based on running time, accuracy.

Chapter 2

Overview

2.1 Test Case Prioritization

Test Case Prioritization is the procedure of ordering the test cases to be executed in a particular order; the test cases are prioritized in higher to lower priority. To increase the test suite rate of fault detection, test cases are prioritized. And also its used to reduce the regression testing's cost. Software tester prioritizes the test case and measure the run of the regression testing process earlier. Information sources of Test case Prioritization are contains the Test management tool, Change management tool, Fault management tool and Architecture models. It's used to prioritize the test cases using Prioritized System.

(Ref: Semantics Scholar)

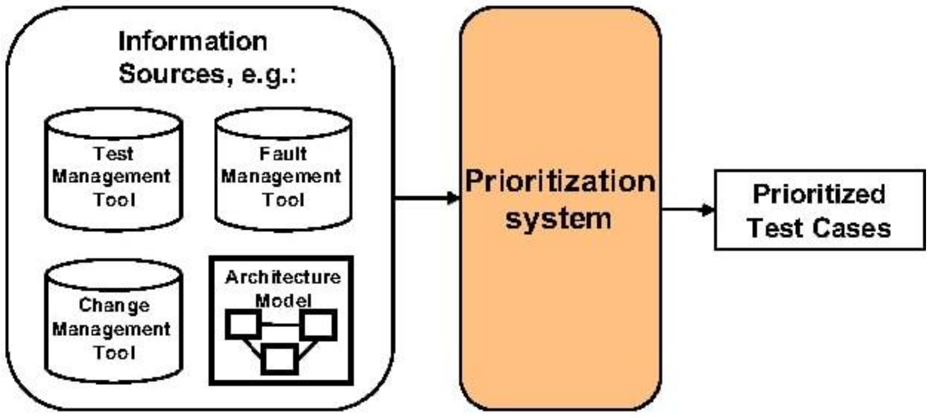


Figure 2.1: Architecture Diagram of Prioritizing Test Case

2.1.1 Literature Review

To solve the problem of test case prioritization, various algorithms such as Search algorithm and Meta-heuristic algorithm are used.

Shruti Mishra et. al. [1] designed the approach of the test-case prioritization by using binary Particle Swarm Optimization Technique. The test case selection and prioritization are used to determine which test cases need to executed thereby making the results more effective.

Manika Tyagi and **Sona Malhotra** [2] proposed an approach based on the Test case Prioritization using Multi-Objective Particle Swarm Optimizer (MOPSO). The goal of regression testing is to validate the modified software. And it has 3 phase approach to solve test-case prioritization. In first phase just remove the redundant test case using simple matrix operation. In second phase, MOPSO used to optimizes the fault coverage and execution time. In third phase, its prioritize the test case from second phase and calculate the ratio of fault coverage to execution time, to check the higher the value of the ratio of higher will be priority the test cases.

Vedpal and **Naresh Chauhan** [3] proposed a test case prioritization technique for object oriented software using method complexity. Minimization of cost and time for testing the software is a challenging task for every software industry. The method complexity is determined by using some factors which are identified by the structural analysis of the software.

Renuka Singh and **Sania Thomas** [4] solved the problem of test case optimization for regression testing bases on hybrid firefly technique. The software presentation phase on a software development life cycle involves regression testing to be done for existing system suite whenever any alterations are done to the software test case optimization in a specification-based on requirement security score.

K.Senthil Kumar and **A.Muthukumaravel** [5] proposed to use a hybrid approach for test case prioritization using PSO based on software quality metrics. Selection of suitable method for optimization and prioritization off any test case as well as appropriate evaluation of the application would result in reduction of fault detection effort without appreciable information loss and significantly decrease the clearing up cost.

Erum Ashraf et. al. [6] designed the technique to check the value based PSO test case prioritization algorithm**.** The aim of the research is to detect the maximum faults earlier in testing life cycle, it have introduced the combination of six prioritization factors for prioritization are customer priority, requirement volatility, implementation complexity, requirement traceability, execution time and fault impact if requirement.

Title	Author	Source
Test-Case Prioritization by using binary Particle Swarm Optimization Technique	Shruti Mishra, Narendra Sharma, Rishi Singh Kushwah	International Journal of Current Trends in Engineering and Technology vol 3, no. 1. Jan 2017
Test case Prioritization using Multi-Objective Particle Swarm Optimizer	Manika Tyagi, Sona Malhotra	International Conference on Signal Propagation and Computer Technology
Test Case Prioritization Technique for Object Oriented Software Using Method Complexity	Vedpal, Naresh Chauhan	International Journal of Innovative Computing, Information and Control, vol 14, no. 1, Feb 2018
Test Case Optimization for Regression Testing Bases on Hybrid Firefly Technique	Renuka Singh, Sania Thomas	International Journal of Advanced Research in Electronics and Communications Engineering vol 6, issue 7, July 2017
A Hybrid Approach for Test Case Prioritization using PSO Based on Software Quality Metrics	K.Senthil Kumar, A.Muthukumaravel	International Journal of Engineering and Technology vol 7, no. 1, Dec 2017
Value based PSO Test Case Prioritization Algorithm	Erum Ashraf, Tamim Ahmed Khan, Khurrum Mahmood, Shaftab Ahmed	International Journal of Advanced Computer Science and Applications vol 8, no.1, 2017

Table 2.1 Literature Survey

2.2 System Planning

Gantt chart

Gantt chart is used to show the timeline of the project work. It specifies the different project module. Each module has a certain time period, the time period can be in the form of the start and the end dates of the respective modules.

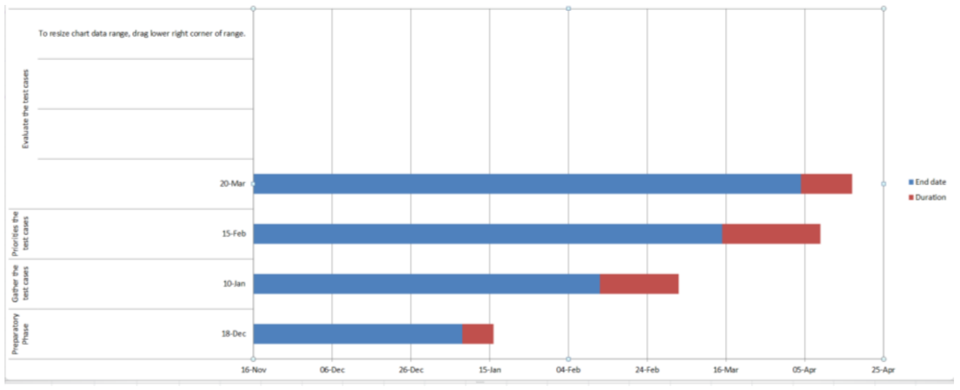


Figure 2.2: Gantt chart

2.4 System Requirements

2.4.1 Hardware Requirements

Processor: Intel Core i5 8th Gen

RAM: 8GB

2.4.2 Software Requirements

Operating System: Windows 10

Browser: Google Chrome

Editor: Jet Brains Pycharm Community Edition,
Command Prompt

Chapter 3

System Design

3.1 Test Case Prioritization (TCP)

3.1 TCP overall architecture

(Ref: Semantics Scholar)

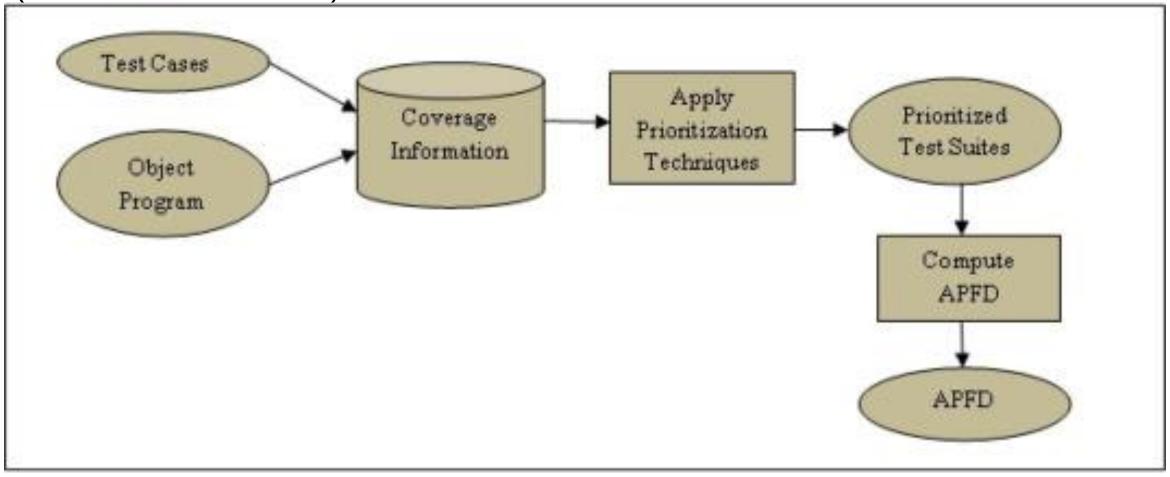


Figure 3.1 TCP Overall Architecture

3.2 Use Case Diagram

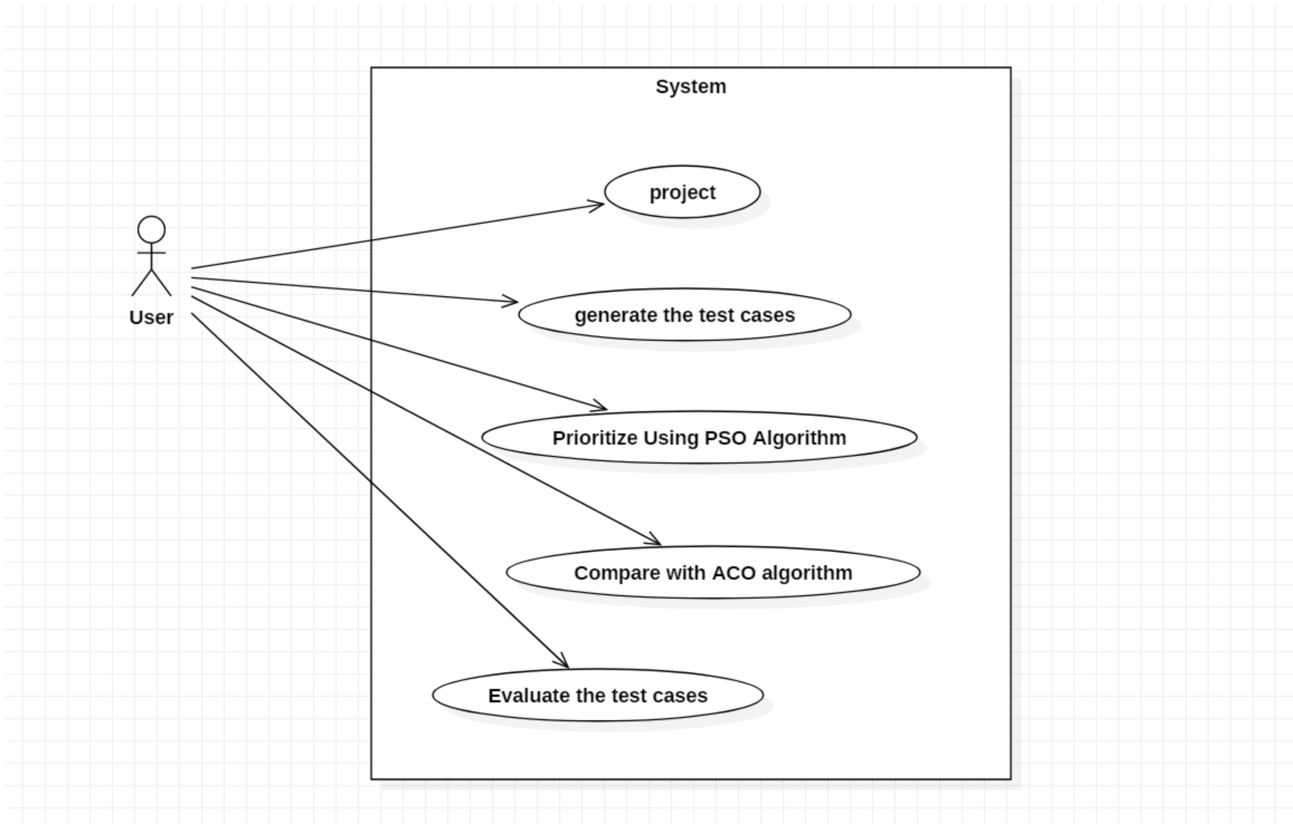


Figure 3.2 Use Case Diagram

3.3 Class Diagram

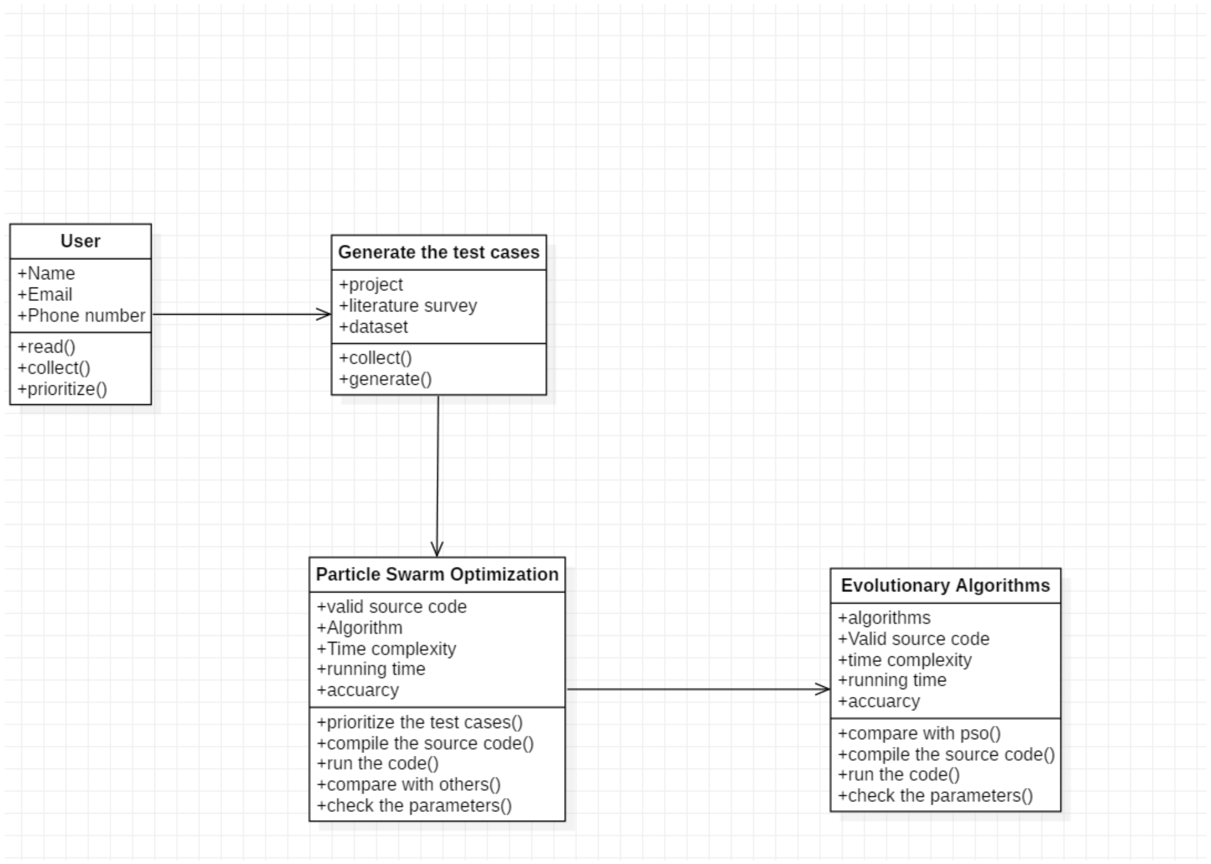


Figure 3.3 Class Diagram

3.4 Sequence Diagram

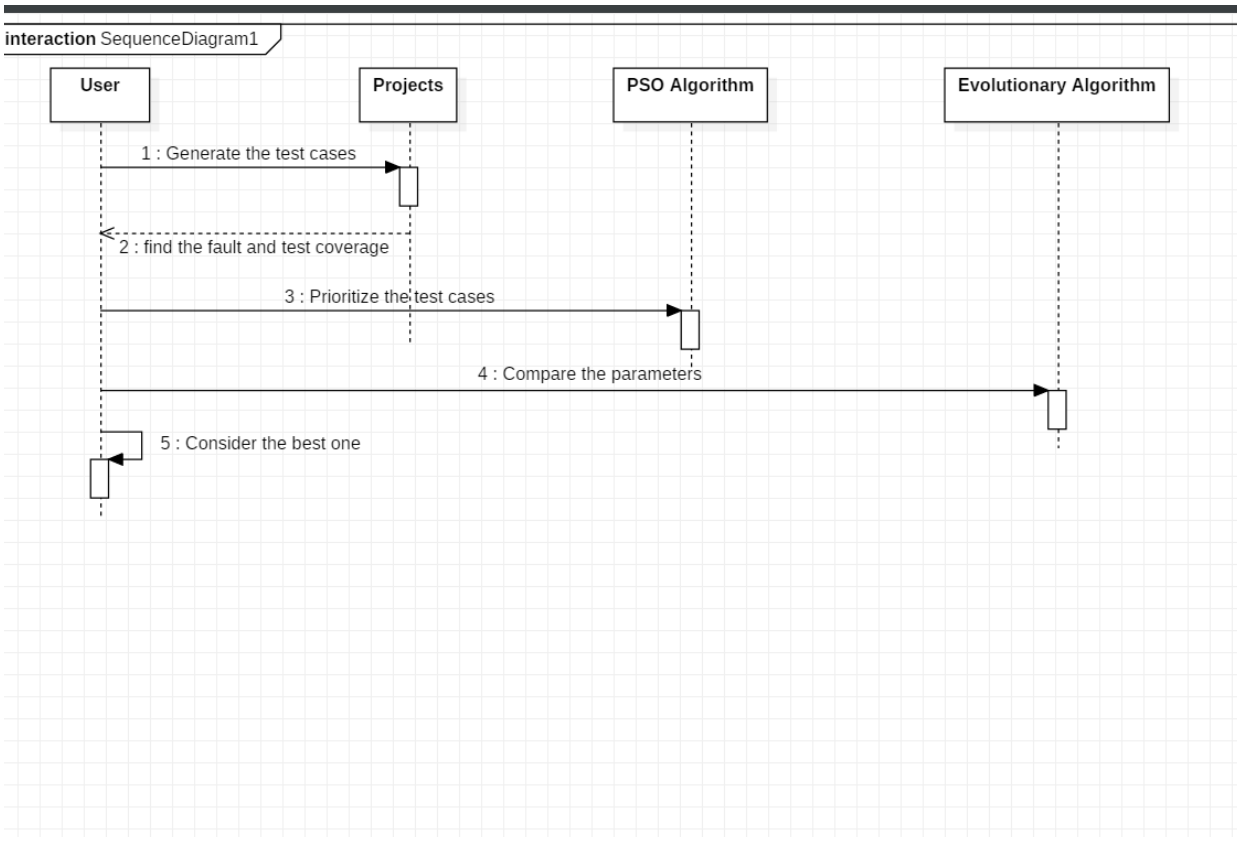

interaction SequenceDiagram1
User
Projects
PSO Algorithm
Evolutionary Algorithm
1 : Generate the test cases
2 : find the fault and test coverage
3 : Prioritize the test cases
4 : Compare the parameters
5 : Consider the best one

Figure 3.4 Sequence Diagram

3.5 Process-Flow Diagram

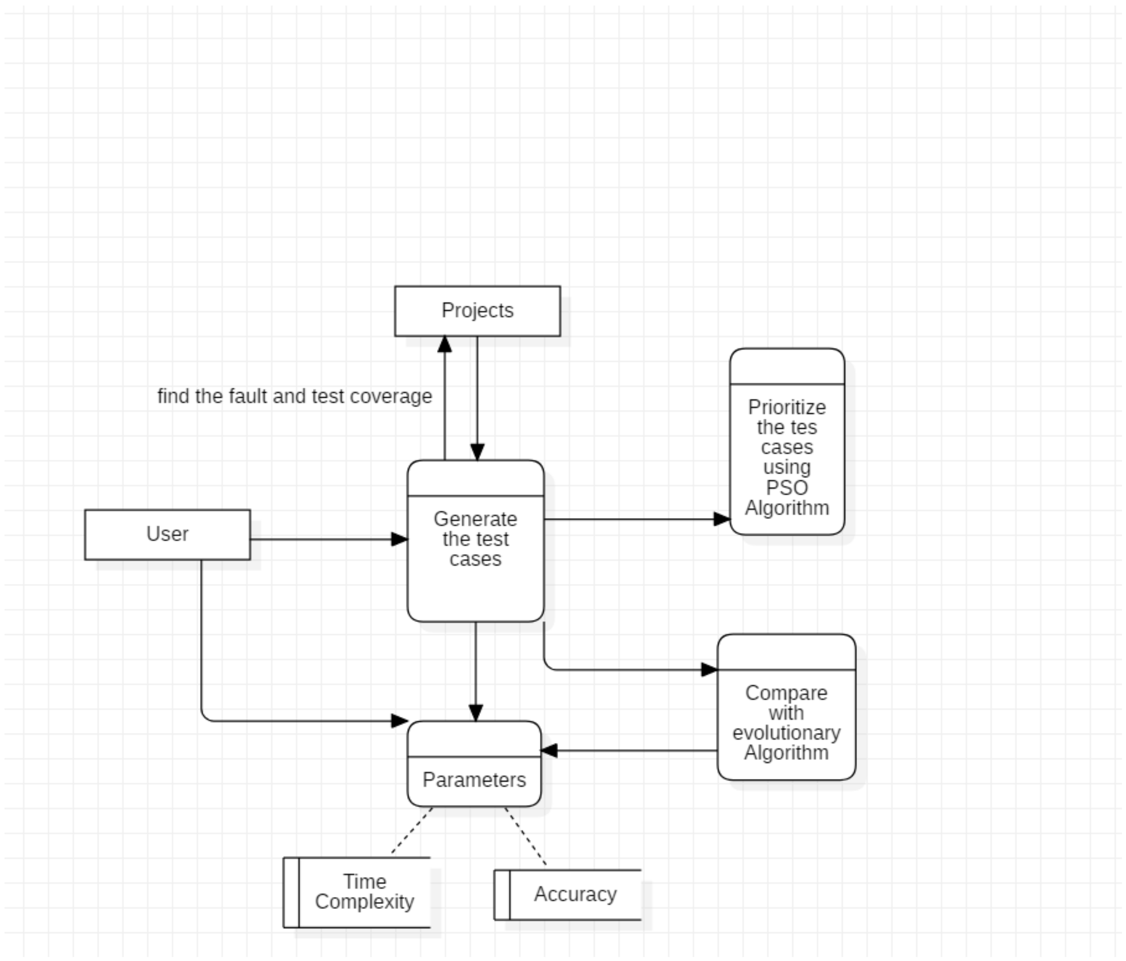

Projects
find the fault and test coverage
Prioritize the tes cases using PSO Algorithm
User
Generate the test cases
Compare with evolutionary Algorithm
Parameters
Time Complexity
Accuracy

Figure 3.5 Process-Flow Diagram

Chapter 4

Implementation of System/ Methodology

This chapter describes about the **Particle Swarm Optimization**. The PSO algorithm is a swarm based-intelligence algorithm. It is inspired by the social activities of insects, bird and fish. Individual ants, bees or birds have limited intelligence when they act alone. But through social interaction with one another as well as their environment, they are capable of accomplishing difficult tasks like discovering the shortest route to a food source, organizing the nest, synchronizing their movement. A particle is similar to a bird or fish, which flying through the problem search space. All the particles movement has been synchronized through a velocity of magnitude and direction. Number of particles present in a problem space is called as population .Each single particle has a heuristic value called solution in search space. All the particles have fitness values which are evaluated by the fitness function to be optimized and have velocities which direct the flying of the particles. The particles fly through the problem space by following the current optimum particles.

PSO is initialized with a group of random particles and then searches for optima by updating generations. In every iteration, each particle is updated by following two best values. The first one is the personal best solution that has achieved so far by neighboring particle. This value is called "pbest".Every time particles based on new pbest. The second is the global best, obtained so far by all particular in the population and called "gbest".The gbest is the actual solution of current iteration

PSO algorithm:-

1. Set particle dimensions equal to the size of ready tasks in {ti} ∊ T,{Ii = 1,2 ,…,n}
2. Initialize particles position randomly from x and velocity v randomly.
3. Calculate its fitness value
4. Compare new fitness value with previous best pbest
5. If the fitness value is better than pbest, set the current fitness value as the pbest.
6. Sort the particles based on fitness, select the best particle as gbest.
7. Update velocity and positions using given formula.
8. If the stopping criteria or maximum iteration is not satisfied, repeat from step3
9. End

.

4.1 Development Tools

4.1.1 Jet Brains PyCharm:

PyCharm is an Integrated Development Environment by Jet brains. It is used for development in Python and frameworks like DJango. It lets you to enhance productivity while coding by providing some features. PyCharm provides API, so the developers can write their own plugins to extend PyCharm features. Several plugins from other Jet Brains IDE also work with PyCharm. There are more than 1000 plugins which are compatible with PyCharm.

- Python:
  Python is a high-level language created by Guido van Rossum and it’s first released in 1991. Python is interpreted, general-purpose programming language. It has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

- Command Prompt:
  Command Prompt is a command line interpreter application available in most Windows operating systems. It's used to execute entered commands. Most of those commands automate tasks via scripts and batch files, perform advanced administrative functions, and troubleshoot or solve certain kinds of Windows issues.

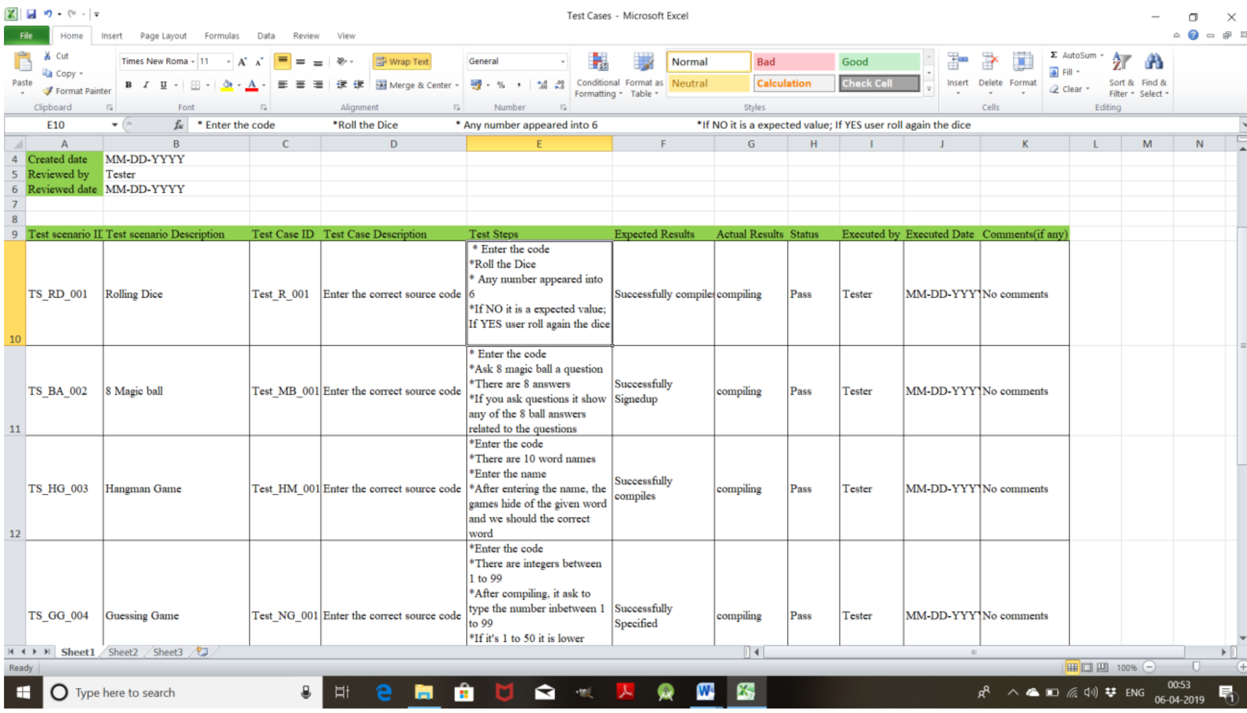
4.2 Testing

4.2.1 Testing

It involves identifying bug/error/defect in software without correcting it. Normally professionals with a quality assurance background are involved in bugs identification. Testing is performed in the testing phase.

4.2.2 Test cases

Games Test cases:



E10 * Enter the code *Roll the Dice * Any number appeared into 6 *If NO it is a expected value; If YES user roll again the dice

Created date	MM-DD-YYYY
Reviewed by	Tester
Reviewed date	MM-DD-YYYY

Test scenario II	Test scenario Description	Test Case ID	Test Case Description	Test Steps	Expected Results	Actual Results	Status	Executed by	Executed Date	Comments(if any)
TS_RD_001	Rolling Dice	Test_R_001	Enter the correct source code	* Enter the code *Roll the Dice * Any number appeared into 6 *If NO it is a expected value; If YES user roll again the dice	Successfully compile	compiling	Pass	Tester	MM-DD-YYY	No comments
TS_BA_002	8 Magic ball	Test_MB_001	Enter the correct source code	* Enter the code *Ask 8 magic ball a question *There are 8 answers *If you ask questions it show any of the 8 ball answers related to the questions	Successfully Signedup	compiling	Pass	Tester	MM-DD-YYY	No comments
TS_HG_003	Hangman Game	Test_HM_001	Enter the correct source code	*Enter the code *There are 10 word names *Enter the name *After entering the name, the games hide of the given word and we should the correct word	Successfully compiles	compiling	Pass	Tester	MM-DD-YYY	No comments
TS_GG_004	Guessing Game	Test_NG_001	Enter the correct source code	*Enter the code *There are integers between 1 to 99 *After compiling, it ask to type the number inbetween 1 to 99 *If it's 1 to 50 it is lower	Successfully Specified	compiling	Pass	Tester	MM-DD-YYY	No comments

Figure 4.1 Screenshot of Generated Test Cases

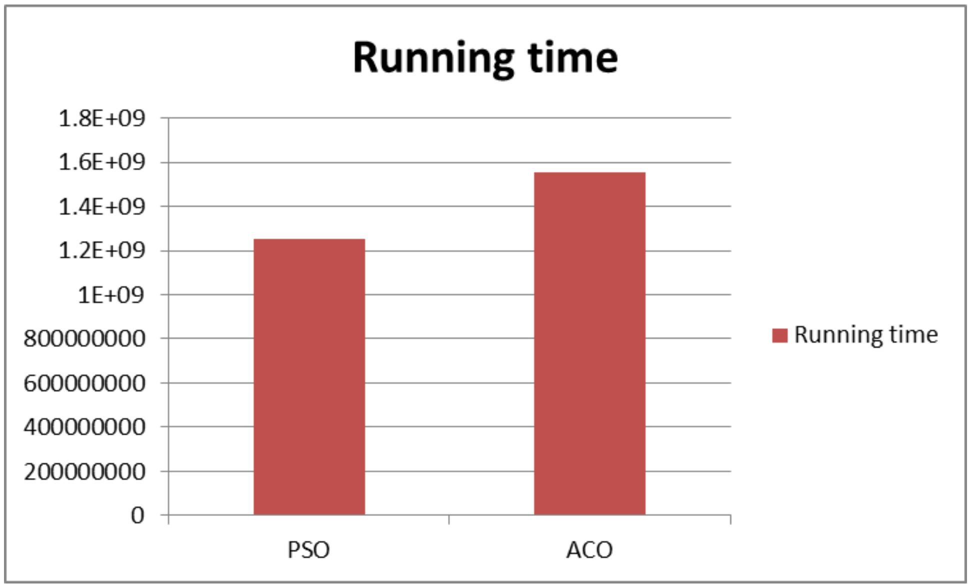
Chapter 5

Results and Discussions

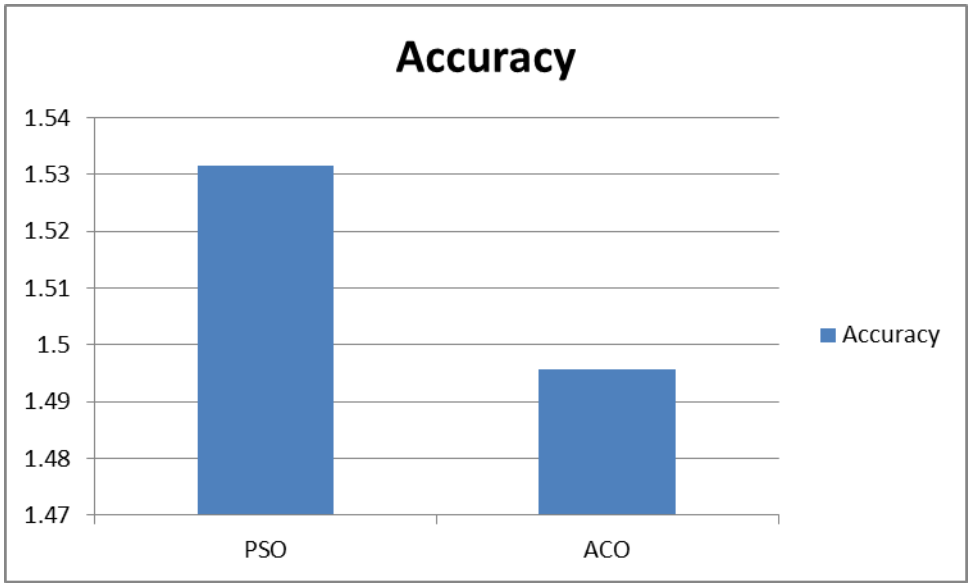
The Test case Prioritization using Meta-heuristic Algorithm is developed in Python. Before prioritization, The test case are generated form the projects like Rolling a dice, 8-Magic Ball, Hangman, Guessing Game; After get the result of projects, find the fault and test coverage to prioritize the test case.

After Prioritization, using Particle Swarm Optimization gets the best iteration of the project using velocity and position of the particles and parameters like time complexity, running time error and accuracy of the code. Then evaluate the algorithm with other evolutionary algorithm to show which one has best results.

When comparing the results, the running time of Particle Swarm Optimization (PSO) is lesser than the Ant Colony Optimization (ACO) for the generated test cases.



When comparing the results, the accuracy of Particle Swarm Optimization (PSO) is higher than the Ant Colony Optimization (ACO) for the generated test cases.



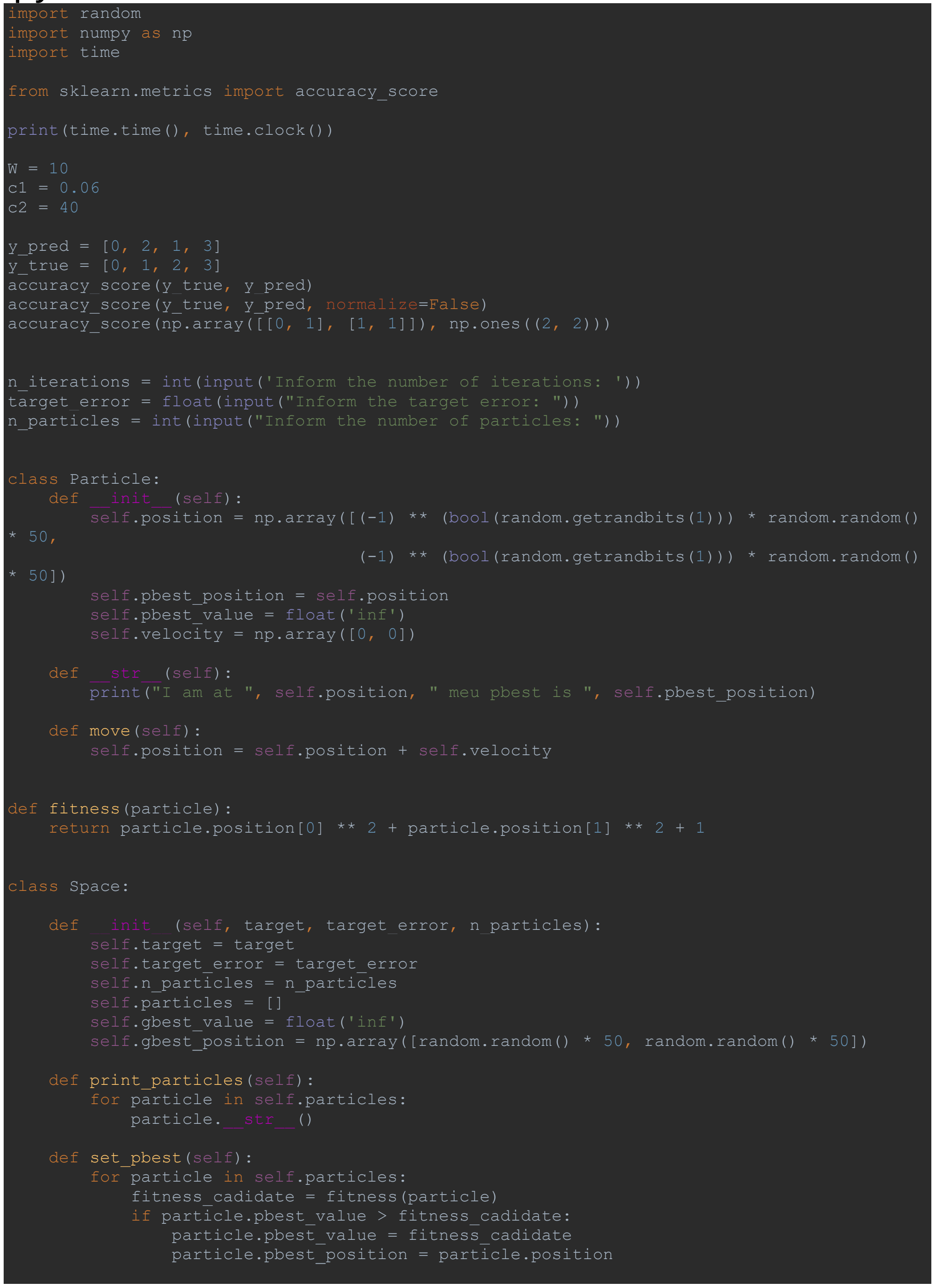
Games	Results
Rolling a dice	• Roll a dice • Number appeared in between 1 to 6 • If we type yes; the dice will roll again • If we type no; that is the expected value
8-Magic Ball	• Ask the question in 8-magic ball • There will be 8 answers • Ask the question it will show the relevant answers in 8-magic ball
Hangman	• The word to guess is in row of dashes • Player guess a letter which exists in word • Player has 10 turns to guess the word
Guessing Game	• The game will add the counter for many guess • Counter initial set from 0 to 99 • User just guess the integer is lower or upper

Table 5.1 Results

5.1 Screen shots

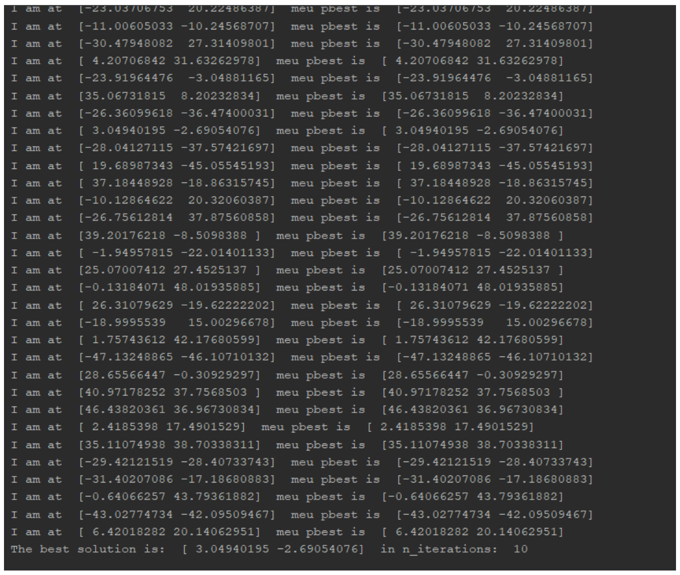
Code:

PSO.py:



```
import random
import numpy as np
import time

from sklearn.metrics import accuracy_score

print(time.time(), time.clock())

W = 10
c1 = 0.06
c2 = 40

y_pred = [0, 2, 1, 3]
y_true = [0, 1, 2, 3]
accuracy_score(y_true, y_pred)
accuracy_score(y_true, y_pred, normalize=False)
accuracy_score(np.array([[0, 1], [1, 1]]), np.ones((2, 2)))


n_iterations = int(input('Inform the number of iterations: '))
target_error = float(input("Inform the target error: "))
n_particles = int(input("Inform the number of particles: "))


class Particle:
    def __init__(self):
        self.position = np.array([(-1) ** (bool(random.getrandbits(1))) * random.random()
* 50,
                                  (-1) ** (bool(random.getrandbits(1))) * random.random()
* 50])
        self.pbest_position = self.position
        self.pbest_value = float('inf')
        self.velocity = np.array([0, 0])

    def __str__(self):
        print("I am at ", self.position, " meu pbest is ", self.pbest_position)

    def move(self):
        self.position = self.position + self.velocity


def fitness(particle):
    return particle.position[0] ** 2 + particle.position[1] ** 2 + 1


class Space:

    def __init__(self, target, target_error, n_particles):
        self.target = target
        self.target_error = target_error
        self.n_particles = n_particles
        self.particles = []
        self.gbest_value = float('inf')
        self.gbest_position = np.array([random.random() * 50, random.random() * 50])

    def print_particles(self):
        for particle in self.particles:
            particle.__str__()

    def set_pbest(self):
        for particle in self.particles:
            fitness_cadidate = fitness(particle)
            if particle.pbest_value > fitness_cadidate:
                particle.pbest_value = fitness_cadidate
                particle.pbest_position = particle.position
    def set_gbest(self):
        for particle in self.particles:
            best_fitness_cadidate = fitness(particle)
            if self.gbest_value > best_fitness_cadidate:
                self.gbest_value = best_fitness_cadidate
                self.gbest_position = particle.position

    def move_particles(self):
        for particle in self.particles:
            global W
            new_velocity = (W * particle.velocity) + (c1 * random.random()) * (
                    particle.pbest_position - particle.position) + \
                           (random.random() * c2) * (self.gbest_position -
particle.position)
            particle.velocity = new_velocity
            particle.move()


search_space = Space(1, target_error, n_particles)
particles_vector = [Particle() for _ in range(search_space.n_particles)]
search_space.particles = particles_vector
search_space.print_particles()

iteration: int = 0
while iteration < n_iterations:
    search_space.set_pbest()
    search_space.set_gbest()

    if (abs(search_space.gbest_value - search_space.target) <=
search_space.target_error):
        break

    search_space.move_particles()
    iteration += 1

print("The best solution is: ", search_space.gbest_position, " in n_iterations: ",
iteration)
```

Output: (Figure 5.1- PSO)



```
I am at  [-23.03706753  20.22486387]  meu pbest is  [-23.03706753  20.22486387]
I am at  [-11.00605033 -10.24568707]  meu pbest is  [-11.00605033 -10.24568707]
I am at  [-30.47948082  27.31409801]  meu pbest is  [-30.47948082  27.31409801]
I am at  [ 4.20706842 31.63262978]  meu pbest is  [ 4.20706842 31.63262978]
I am at  [-23.91964476  -3.04881165]  meu pbest is  [-23.91964476  -3.04881165]
I am at  [35.06731815  8.20232834]  meu pbest is  [35.06731815  8.20232834]
I am at  [-26.36099618 -36.47400031]  meu pbest is  [-26.36099618 -36.47400031]
I am at  [ 3.04940195 -2.69054076]  meu pbest is  [ 3.04940195 -2.69054076]
I am at  [-28.04127115 -37.57421697]  meu pbest is  [-28.04127115 -37.57421697]
I am at  [ 19.68987343 -45.05545193]  meu pbest is  [ 19.68987343 -45.05545193]
I am at  [ 37.18448928 -18.86315745]  meu pbest is  [ 37.18448928 -18.86315745]
I am at  [-10.12864622  20.32060387]  meu pbest is  [-10.12864622  20.32060387]
I am at  [-26.75612814  37.87560858]  meu pbest is  [-26.75612814  37.87560858]
I am at  [39.20176218 -8.5098388 ]  meu pbest is  [39.20176218 -8.5098388 ]
I am at  [ -1.94957815 -22.01401133]  meu pbest is  [ -1.94957815 -22.01401133]
I am at  [25.07007412 27.4525137 ]  meu pbest is  [25.07007412 27.4525137 ]
I am at  [-0.13184071 48.01935885]  meu pbest is  [-0.13184071 48.01935885]
I am at  [ 26.31079629 -19.62222202]  meu pbest is  [ 26.31079629 -19.62222202]
I am at  [-18.9995539   15.00296678]  meu pbest is  [-18.9995539   15.00296678]
I am at  [ 1.75743612 42.17680599]  meu pbest is  [ 1.75743612 42.17680599]
I am at  [-47.13248865 -46.10710132]  meu pbest is  [-47.13248865 -46.10710132]
I am at  [28.65566447 -0.30929297]  meu pbest is  [28.65566447 -0.30929297]
I am at  [40.97178252 37.7568503 ]  meu pbest is  [40.97178252 37.7568503 ]
I am at  [46.43820361 36.96730834]  meu pbest is  [46.43820361 36.96730834]
I am at  [ 2.4185398 17.4901529]  meu pbest is  [ 2.4185398 17.4901529]
I am at  [35.11074938 38.70338311]  meu pbest is  [35.11074938 38.70338311]
I am at  [-29.42121519 -28.40733743]  meu pbest is  [-29.42121519 -28.40733743]
I am at  [-31.40207086 -17.18680883]  meu pbest is  [-31.40207086 -17.18680883]
I am at  [-0.64066257 43.79361882]  meu pbest is  [-0.64066257 43.79361882]
I am at  [-43.02774734 -42.09509467]  meu pbest is  [-43.02774734 -42.09509467]
I am at  [ 6.42018282 20.14062951]  meu pbest is  [ 6.42018282 20.14062951]
The best solution is:  [ 3.04940195 -2.69054076]  in n_iterations:  10
```

Running time:

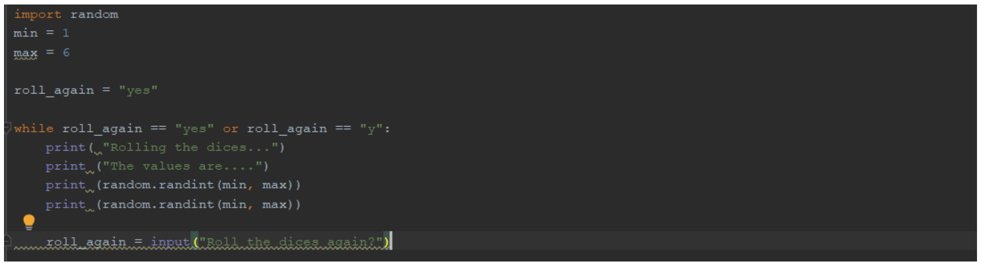
```
C:\Users\Lenovo\PycharmProjects\untitled\venv\Scripts\python.exe
C:/Users/Lenovo/PycharmProjects/face/pso.py
1253348879.8392978
```

Accuracy:

```
C:\Users\Lenovo\PycharmProjects\untitled\venv\Scripts\python.exe
C:/Users/Lenovo/PycharmProjects/face/pso.py
 1.531557718
```

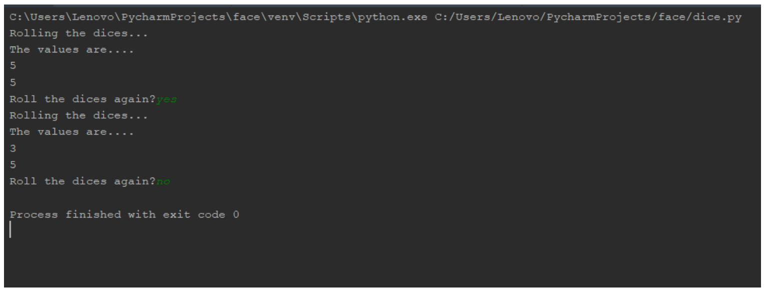
Rolling a Dice

Code:

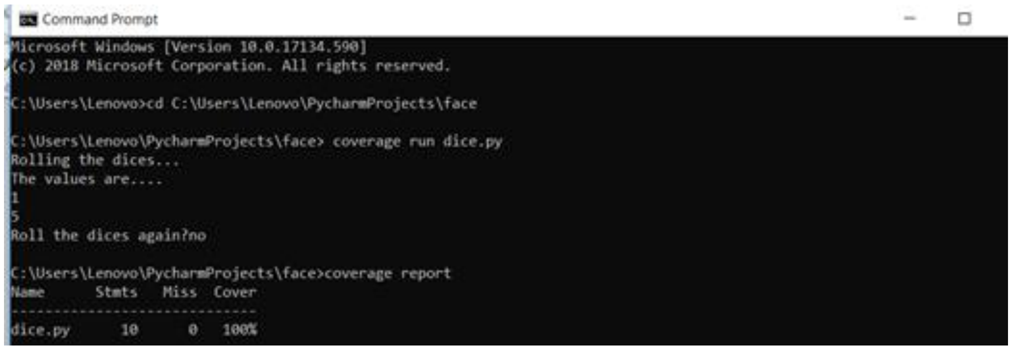


```
import random
min = 1
max = 6

roll_again = "yes"

while roll_again == "yes" or roll_again == "y":
    print ("Rolling the dices...")
    print ("The values are....")
    print (random.randint(min, max))
    print (random.randint(min, max))

    roll_again = input("Roll the dices again?")
```

Output:



```
C:\Users\Lenovo\PycharmProjects\face\venv\Scripts\python.exe C:/Users/Lenovo/PycharmProjects/face/dice.py
Rolling the dices...
The values are....
5
5
Roll the dices again?yes
Rolling the dices...
The values are....
3
5
Roll the dices again?no

Process finished with exit code 0
```

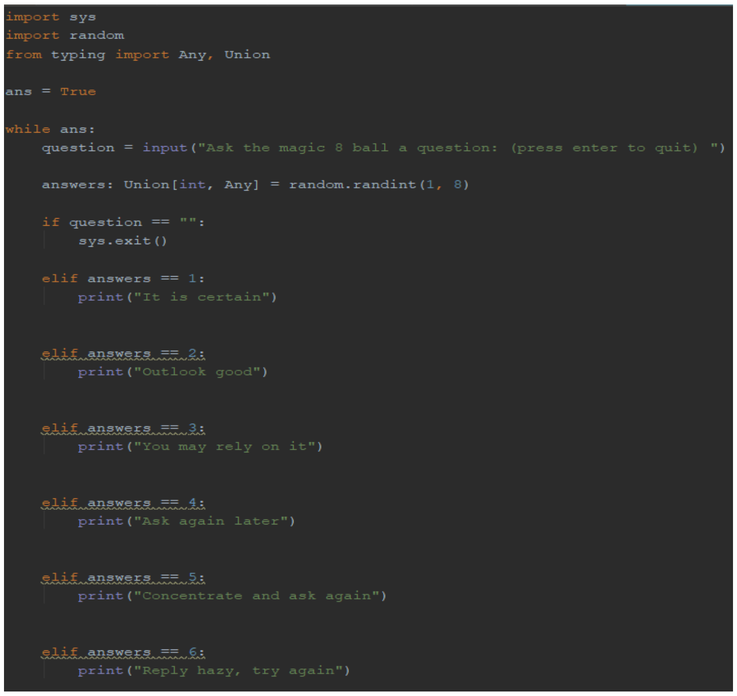


```
Microsoft Windows [Version 10.0.17134.590]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Lenovo>cd C:\Users\Lenovo\PycharmProjects\face

C:\Users\Lenovo\PycharmProjects\face> coverage run dice.py
Rolling the dices...
The values are....
1
5
Roll the dices again?no

C:\Users\Lenovo\PycharmProjects\face>coverage report
Name      Stmts   Miss  Cover
-----------------------------
dice.py      10      0   100%
```

\

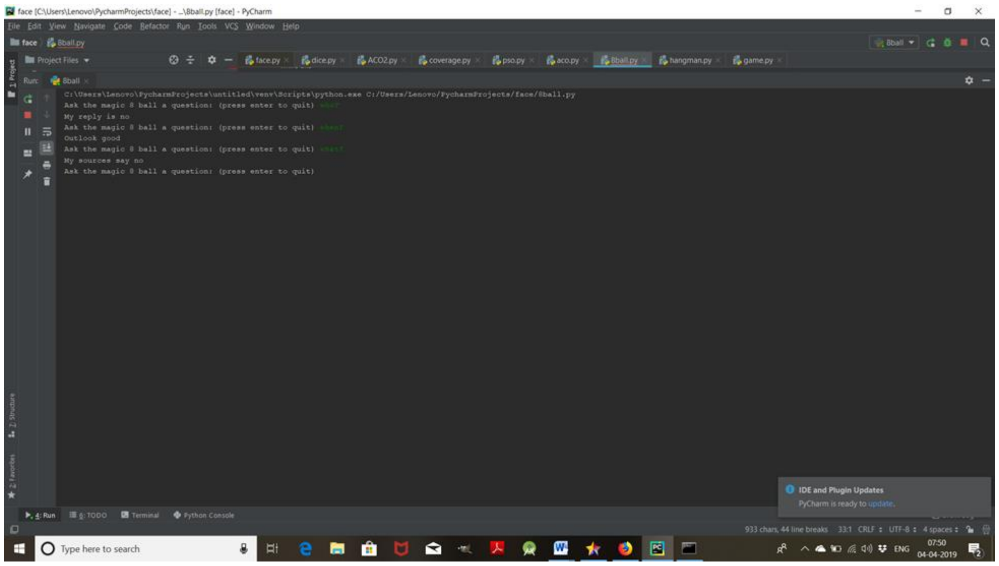
Figure 5.3 Test coverage-1

8-Magic Ball:



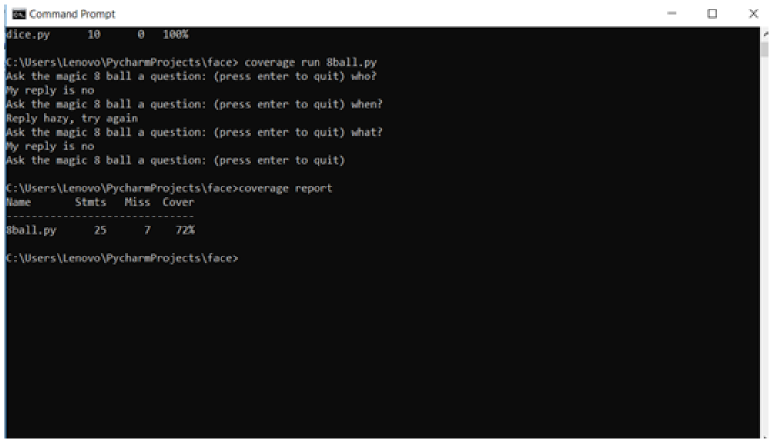
```python
import sys
import random
from typing import Any, Union

ans = True

while ans:
    question = input("Ask the magic 8 ball a question: (press enter to quit) ")

    answers: Union[int, Any] = random.randint(1, 8)

    if question == "":
        sys.exit()

    elif answers == 1:
        print("It is certain")


    elif answers == 2:
        print("Outlook good")


    elif answers == 3:
        print("You may rely on it")


    elif answers == 4:
        print("Ask again later")


    elif answers == 5:
        print("Concentrate and ask again")


    elif answers == 6:
        print("Reply hazy, try again")
```

Output:



```
C:\Users\Lenovo\PycharmProjects\untitled\venv\Scripts\python.exe C:/Users/Lenovo/PycharmProjects/face/8ball.py
Ask the magic 8 ball a question: (press enter to quit) what?
My reply is no
Ask the magic 8 ball a question: (press enter to quit) when?
Outlook good
Ask the magic 8 ball a question: (press enter to quit) what?
My sources say no
Ask the magic 8 ball a question: (press enter to quit)
```

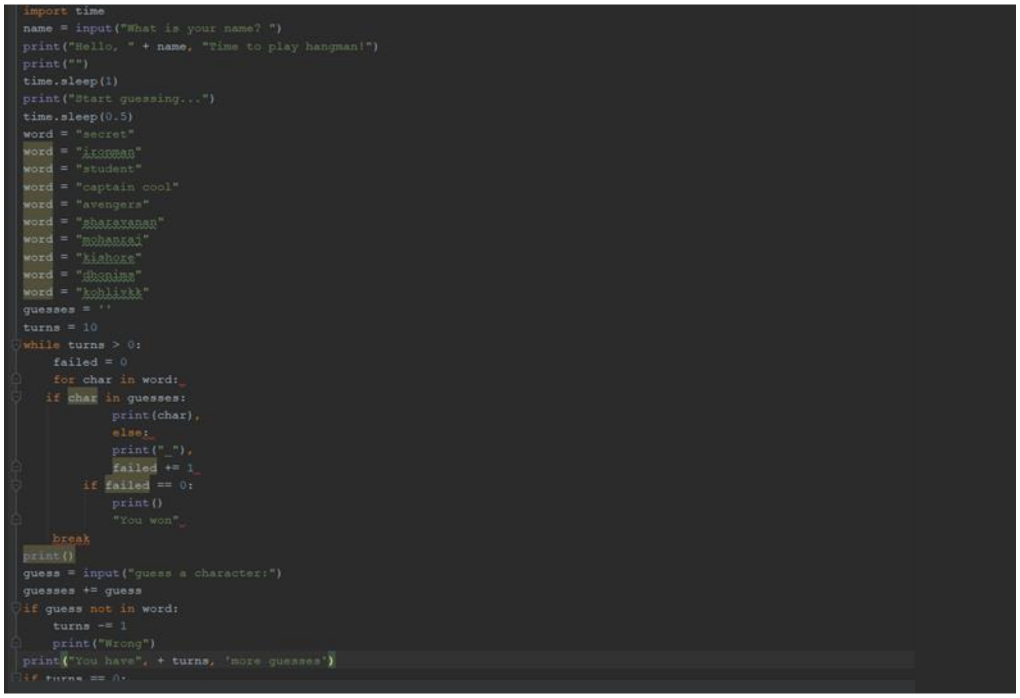
Figure 5.4 8-ball



```
dice.py      10      0   100%

C:\Users\Lenovo\PycharmProjects\face> coverage run 8ball.py
Ask the magic 8 ball a question: (press enter to quit) who?
My reply is no
Ask the magic 8 ball a question: (press enter to quit) when?
Reply hazy, try again
Ask the magic 8 ball a question: (press enter to quit) what?
My reply is no
Ask the magic 8 ball a question: (press enter to quit)

C:\Users\Lenovo\PycharmProjects\face>coverage report
Name       Stmts   Miss  Cover
------------------------------
8ball.py      25      7    72%

C:\Users\Lenovo\PycharmProjects\face>
```

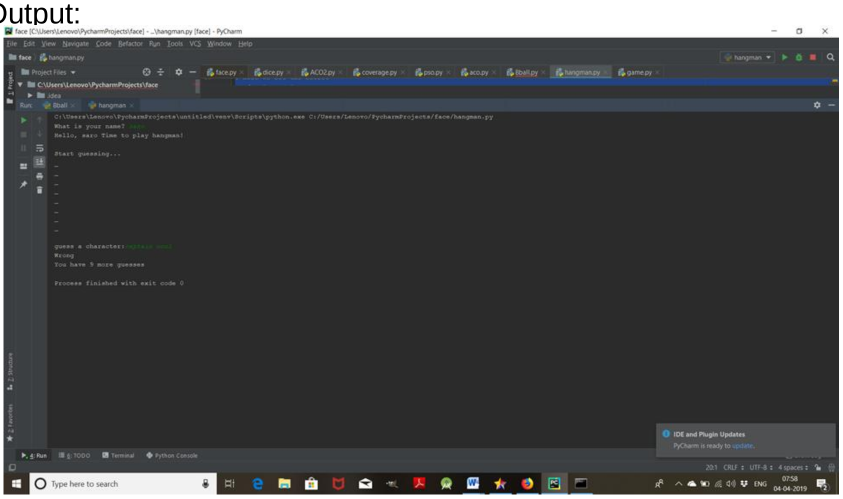
Figure 5.5 Test Coverage-2

Hangman



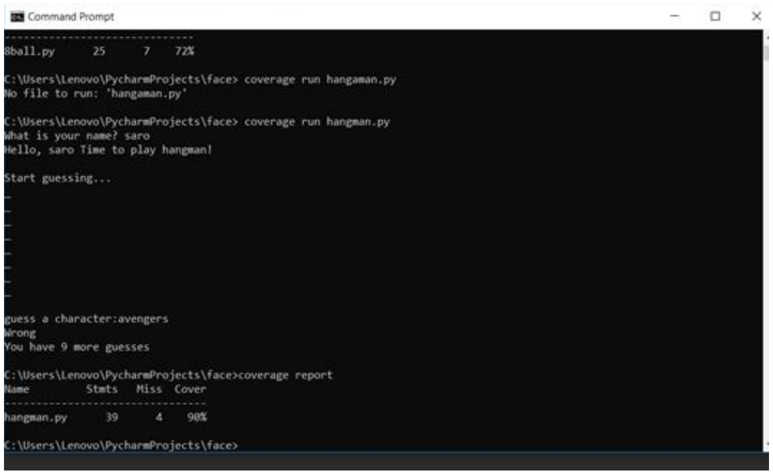
```
import time
name = input("What is your name? ")
print("Hello, " + name, "Time to play hangman!")
print("")
time.sleep(1)
print("Start guessing...")
time.sleep(0.5)
word = "secret"
word = "ironman"
word = "student"
word = "captain cool"
word = "avengers"
word = "sharavanan"
word = "mohanraj"
word = "kishore"
word = "dhonims"
word = "kohlivkk"
guesses = ''
turns = 10
while turns > 0:
    failed = 0
    for char in word:
   if char in guesses:
            print(char),
            else:
            print("_"),
            failed += 1
        if failed == 0:
            print()
            "You won"
    break
print()
guess = input("guess a character:")
guesses += guess
if guess not in word:
    turns -= 1
    print("Wrong")
print("You have", + turns, 'more guesses')
if turns == 0:
```

Output:



```
C:\Users\Lenovo\PycharmProjects\untitled\venv\Scripts\python.exe C:/Users/Lenovo/PycharmProjects/face/hangman.py
What is your name? saro
Hello, saro Time to play hangman!

Start guessing...
_
_
_
_
_
_
_
_

guess a character:captain marvel
Wrong
You have 9 more guesses

Process finished with exit code 0
```

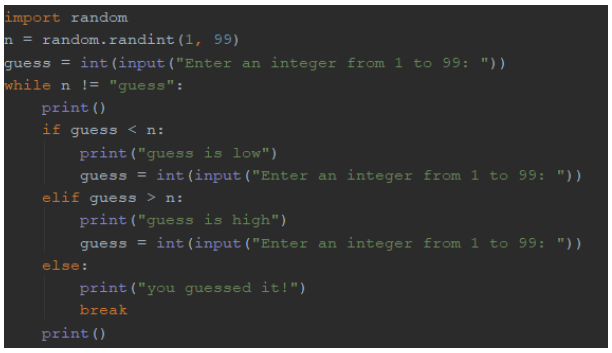
Figure 5.6 Hangman



```
--------------------------------
8ball.py      25      7    72%

C:\Users\Lenovo\PycharmProjects\face> coverage run hangaman.py
No file to run: 'hangaman.py'

C:\Users\Lenovo\PycharmProjects\face> coverage run hangman.py
What is your name? saro
Hello, saro Time to play hangman!

Start guessing...
_
_
_
_
_
_
_
_

guess a character:avengers
Wrong
You have 9 more guesses

C:\Users\Lenovo\PycharmProjects\face>coverage report
Name         Stmts   Miss  Cover
--------------------------------
hangman.py      39      4    90%

C:\Users\Lenovo\PycharmProjects\face>
```

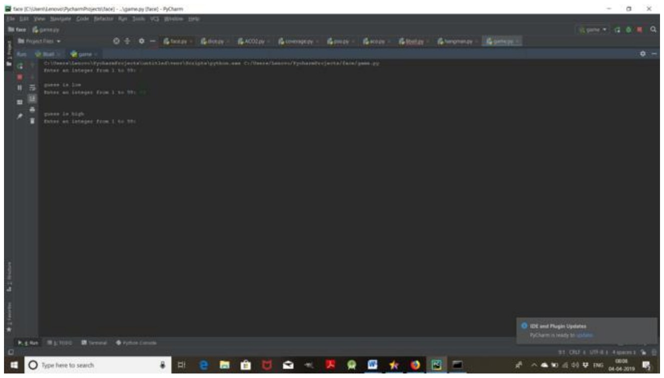
Figure 5.7 Test Coverage-3

Guessing Game



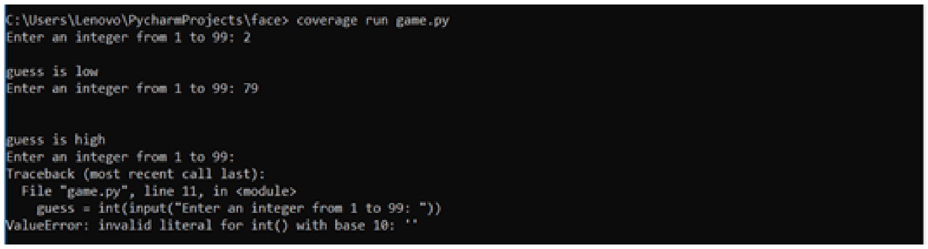
```
import random
n = random.randint(1, 99)
guess = int(input("Enter an integer from 1 to 99: "))
while n != "guess":
    print()
    if guess < n:
        print("guess is low")
        guess = int(input("Enter an integer from 1 to 99: "))
    elif guess > n:
        print("guess is high")
        guess = int(input("Enter an integer from 1 to 99: "))
    else:
        print("you guessed it!")
        break
    print()
```

Output:



```
C:\Users\Lenovo\PycharmProjects\untitled\venv\Scripts\python.exe C:/Users/Lenovo/PycharmProjects/face/game.py
Enter an integer from 1 to 99:

guess is low
Enter an integer from 1 to 99:


guess is high
Enter an integer from 1 to 99:
```

Figure 5.8 Guessing game



```
C:\Users\Lenovo\PycharmProjects\face> coverage run game.py
Enter an integer from 1 to 99: 2

guess is low
Enter an integer from 1 to 99: 79


guess is high
Enter an integer from 1 to 99:
Traceback (most recent call last):
  File "game.py", line 11, in <module>
    guess = int(input("Enter an integer from 1 to 99: "))
ValueError: invalid literal for int() with base 10: ''
```

Figure 5.9 Test Coverage-4

Test Coverage:

Games	Test Coverage
Rolling a dice	C:\Users\Lenovo\PycharmProjects\face> coverage report Name Stmts Miss Cover ----------------------------- dice.py 10 0 100%
8-Magic Ball	C:\Users\Lenovo\PycharmProjects\face> coverage report Name Stmts Miss Cover ------------------------------ 8ball.py 25 7 72%
Hangman	C:\Users\Lenovo\PycharmProjects\face> coverage report Name Stmts Miss Cover -------------------------------- hangman.py 39 4 90%
Guessing Game	C:\Users\Lenovo\PycharmProjects\face> coverage report Name Stmts Miss Cover ----------------------------- game.py 14 2 86%

Table 5.2 Test Coverage

Games	Prioritized test cases
Rolling a dice	100%
Hangman	90%
Guessing Game	86%
8-Magic ball	72%

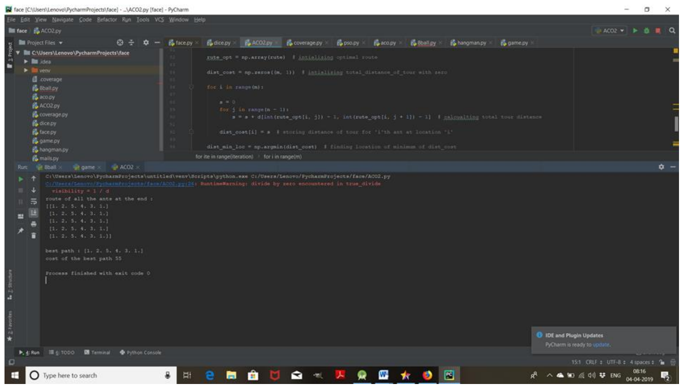
Table 5.3 Prioritized Results

ACO Code:

```
                    import numpy as np
from numpy import inf
import time

from sklearn.metrics import accuracy_score

print(time.time(), time.clock())

# given values for the problems

d = np.array([[0, 10, 12, 11, 14]
        , [10, 0, 13, 15, 8]
        , [12, 13, 0, 9, 14]
        , [11, 15, 9, 0, 16]
        , [14, 8, 14, 16, 0]])

iteration = 100
n_ants = 5
n_citys = 5
y_pred = [0, 2, 1, 3]
y_true = [0, 1, 2, 3]
accuracy_score(y_true, y_pred)
accuracy_score(y_true, y_pred, normalize=False)
accuracy_score(np.array([[0, 1], [1, 1]]), np.ones((2, 2)))

# intialization part

m = n_ants
n = n_citys
e = .5  # evaporation rate
alpha = 1  # pheromone factor
beta = 2  # visibility factor

# calculating the visibility of the next city visibility(i,j)=1/d(i,j)

visibility = 1 / d
visibility[visibility == inf] = 0

# intializing pheromne present at the paths to the cities

pheromne = .1 * np.ones((m, n))

# intializing the rute of the ants with size rute(n_ants,n_citys+1)
# note adding 1 because we want to come back to the source city

rute = np.ones((m, n + 1))

for ite in range(iteration):

    rute[:, 0] = 1  # initial starting and ending positon of every ants '1' i.e city '1'

    for i in range(m):

        temp_visibility = np.array(visibility)  # creating a copy of visibility

        for j in range(n - 1):
            # print(rute)

            combine_feature = np.zeros(5)  # intializing combine_feature array to zero
            cum_prob = np.zeros(5)  # intializing cummulative probability array to zeros

            cur_loc = int(rute[i, j] - 1)  # current city of the ant

            temp_visibility[:, cur_loc] = 0  # making visibility of the current city as zero

            p_feature = np.power(pheromne[cur_loc, :], beta)  # calculating pheromne feature
            v_feature = np.power(temp_visibility[cur_loc, :], alpha)  # calculating visibility feature

            p_feature = p_feature[:, np.newaxis]  # adding axis to make a size[5,1]
            v_feature = v_feature[:, np.newaxis]  # adding axis to make a size[5,1]

            combine_feature = np.multiply(p_feature, v_feature)  # calculating the combine feature

            total = np.sum(combine_feature)  # sum of all the feature

            probs = combine_feature / total  # finding probability of element probs(i) = comine_feature(i)/total

            cum_prob = np.cumsum(probs)  # calculating cummulative sum
            # print(cum_prob)
            r = np.random.random_sample()  # randon no in [0,1)
            # print(r)
            city = np.nonzero(cum_prob > r)[0][0] + 1  # finding the next city having probability higher then random(r)
            # print(city)

            rute[i, j + 1] = city  # adding city to route

        left = list(set([i for i in range(1, n + 1)]) - set(rute[i, :-2]))[
            0]  # finding the last untraversed city to route

        rute[i, -2] = left  # adding untraversed city to route

    rute_opt = np.array(rute)  # intializing optimal route

    dist_cost = np.zeros((m, 1))  # intializing total_distance_of_tour with zero

    for i in range(m):

        s = 0
        for j in range(n - 1):
            s = s + d[int(rute_opt[i, j]) - 1, int(rute_opt[i, j + 1]) - 1]  # calcualting total tour distance

        dist_cost[i] = s  # storing distance of tour for 'i'th ant at location 'i'

    dist_min_loc = np.argmin(dist_cost)  # finding location of minimum of dist_cost
```

```
dist_min_cost = dist_cost[dist_min_loc]  # finging min of dist_cost


    best_route = rute[dist_min_loc, :]  # intializing current traversed as best route
    pheromne = (1 - e) * pheromne  # evaporation of pheromne with (1-e)

    for i in range(m):
        for j in range(n - 1):
            dt = 1 / dist_cost[i]
            pheromne[int(rute_opt[i, j]) - 1, int(rute_opt[i, j + 1]) - 1] = pheromne[int(rute_opt[i, j]) - 1, int(
                rute_opt[i, j + 1]) - 1] + dt
            # updating the pheromne with delta_distance
            # delta_distance will be more with min_dist i.e adding more weight to that route  peromne

print('route of all the ants at the end :')
print(rute_opt)
print()
print('best path :', best_route)
print('cost of the best path', int(dist_min_cost[0]) + d[int(best_route[-2]) - 1, 0])
```

Output:



```
    rute_opt = np.array(rute)  # intializing optimal route

    dist_cost = np.zeros((m, 1))  # intializing total_distance_of_tour with zero

    for i in range(m):

        s = 0
        for j in range(n - 1):
            s = s + d[int(rute_opt[i, j]) - 1, int(rute_opt[i, j + 1]) - 1]  # calcualting total tour distance

        dist_cost[i] = s  # storing distance of tour for 'i'th ant at location 'i'

    dist_min_loc = np.argmin(dist_cost)  # finding location of minimum of dist_cost
```

```
C:\Users\Lenovo\PycharmProjects\untitled\venv\Scripts\python.exe C:/Users/Lenovo/PycharmProjects/face/ACO2.py
C:/Users/Lenovo/PycharmProjects/face/ACO2.py:26: RuntimeWarning: divide by zero encountered in true_divide
  visibility = 1 / d
route of all the ants at the end :
[[1. 2. 5. 4. 3. 1.]
 [1. 2. 5. 4. 3. 1.]
 [1. 2. 5. 4. 3. 1.]
 [1. 2. 5. 4. 3. 1.]
 [1. 2. 5. 4. 3. 1.]]

best path : [1. 2. 5. 4. 3. 1.]
cost of the best path 55

Process finished with exit code 0
```

Figure 6.0 ACO

Running Time:

```
C:\Users\Lenovo\PycharmProjects\untitled\venv\Scripts\python.exe
C:/Users/Lenovo/PycharmProjects/face/ACO2.py
1555350090.5324647
```

Accuracy:

```
C:\Users\Lenovo\PycharmProjects\untitled\venv\Scripts\python.exe
C:/Users/Lenovo/PycharmProjects/face/ACO2.py
1.495790794
```

Chapter 6

Conclusion

In the study, the test case dataset for the chosen project or selected for the test case prioritization problem particle swarm optimization (PSO) algorithm has been implemented. Further, the results are analysed with Ant Colony Algorithm (ACO) algorithm for time complexity, running time and accuracy. The evolution proves that PSO out-performs ACO on all the three selected criteria

6.1 References

- Shruti Mishra; Kailash Patidar, 'Test-Case Prioritization using Binary Particle Swarm Optimization Method', International Journal of Current Trends in Engineering and Technology vol 3, no. 1. Jan 2017

- Manika Tyagi, Sona Malhotra, 'Test case Prioritization using Multi-Objective Particle Swarm Optimizer', International Conference on signal Propagation and Computer Technology.

- Vedpal, Naresh Chauhan, ' Test case Prioritization Technique for Object Oriented Software using Method Complexity', International Journal of Innovative Computing, Information and Control, vol 14, no.1, Feb 2018.

- Renuka Singh, Sania Thomas, 'Test case Optimization for Regression Testing Bases on Hybrid Firefly Technique', International Journal of Advanced Research and Communications, Engineering', vol 6, Issue 7, July 2017

- K.Senthil Kumar, A.Muthukumaravel, 'A Hybrid Approach for Test Case Prioritization using PSO Based on Software Quality Metrics', International Journal of Engineering And Technology', vol 7, no.1, Dec 2017

- Dima Suleiman; Marwah Alian; Amjad Hudaib, 'A survey on prioritization regression testing test case', 8th International Conference on Information Technology, Jordan. May 2017.

- Dr.K.Alagarsamy; SriVidhya, 'A synthesized overview of test case optimization techniques', Journal of recent research in engineering and technology vol 1, no.2. June 2014.

- Alireza Khalilian, Mohammed Abdollahi Azgomi, Yalda Fazlalizadeh, 'An improved method for test case prioritization by incorporating historical test case data', School of Computing Engineering, Iran University of Science and Technology, Tehran, Iran.