Journal of Engineering and Applied Sciences 14 (5): 1590-1600, 2019
ISSN: 1816-949X

Automatic Generation and Optimization of Test Cases Using Genetic Algorithm with UML Diagram

Anju Bala and Rajender Singh Chhillar
Department of Computer Science and Applications, Maharshi Dayanand University, Rohtak, India

Abstract: Selection of test case is a standard testing technique to opt a subset of existing test cases for execution, due to the limited budget and other necessary constraints. The key objective of this study is automatic generation and optimization of test cases using bio-inspired Genetic Algorithm (GA). These search optimization techniques lead to global best solution. These algorithms are used to generate test paths and then optimize them. The case study on telemedicine simulation system is being presented here using use case diagrams, activity diagram and sequence diagram. Activity diagram graph and sequence diagram graph show test paths which are being optimized using Genetic algorithm. This study presents a novel approach for generation of test cases using UML. Our approach consists of converting the all UML diagrams into graph and integrated to form System Under Test (SUT). From the graphs different control flow series also called test cases are recognized and then optimized using Genetic algorithm. The system graph is then traversed to generate test paths which are being optimized using GA. To explore the efficacy of our approach, we performed an empirical study using MATLAB programs with manifold paths and other parameters. Our results indicate that generation and optimization of test case is achieved efficiently in much less time.

Key words: Telemedicine diagnosis system, Genetic algorithm, use case, sequence diagram, activity diagram, MATLAB

INTRODUCTION

Testing is both technically and economically an important part of high quality software production. It has been estimated that testing accounts for half of the expenses in software production. Much of the testing is done manually or using other labor-intensive methods. It is thus, vital for the software industry to develop efficient, cost effective and automatic means and tools for software testing. Researchers have proposed several methods over years to generate automatically solution which have different drawbacks. This study examines automatic generation and optimization of test cases using UML (Unified Modeling Language) diagrams by applying genetic algorithm approaches. UML is a standard visual modeling language intended to be used for modeling, analysis, design and implementation of software based systems. UML is a common language for business analysts, software architects and developers. It is used to describe, specify, structure, document and design, existing or new software systems. According to dynamic technology, software systems and their environments change constantly. They are improved, corrected and ported to new platforms to achieve the benchmark of recent time. These changes can affect a system adversely. UML Models are considered to reduce the complexity of the problem with the increase in sizes and changes in environment. Unified modeling language has become the de facto standard for modeling and design of software. It is widely accepted and used by industry.

A sequence diagram explains how objects correspond with each other in terms of a order of messages. A sequence diagram is a best approach to visualize and validate various runtime scenarios. A sequence diagram shows how processes interact with one another and in what order. Activity diagram is another important diagram in UML to describe the dynamic aspects of the system. Other diagrams are used to show the message flow from one object to another but activity diagram is used to show message flow from one activity to another.

Our approach consists of converting the activity diagram into Activity Graph (AG) and sequence diagram into Sequence Graph (SG). Then, these graphs are integrated to form System Under Test (SUT). This study presents a novel approach for generation of test cases using UML. Simulated telemedicine software is taken as case study which is used to connect with patients with

Corresponding Author: Anju Bala, Department of Computer Science and Applications, Maharshi Dayanand University, Rohtak, India

least cost, accessible to consultants from specialists with increased patient engagement and better patient care. Activity graph and sequence graph show test paths which are being optimized using Genetic algorithm. Genetic algorithm is heuristic based search method by exploring good multidimensional search by maintaining an optimized population, random actions consisting of the combination regarding iterative search steps.

Literature review: A lot of research has been done in the field of test data generation. The major factors involved in test case generation are UML diagrams, different testing types, different testing tools, different optimization search techniques and various other approaches.

A technique proposed by Sabharwal *et al*. (2010) for the prior of test cases scenario which are derived from state chart and activity diagram usinf Genetic algorithm and stack information flow diagram. Stack based application is used for allocating weight to each node of state and activity diagram. Sumalatha and Raju (2013) presents a test case generation with UML's sequence diagram by using Genetic algorithm where test cases are optimized. By applying Genetic algorithm on sequence graph, starting from the source, all paths are discovered up to destination with loops also calculating the fitness value for these paths. Maheshwari and Prasanna (2015) have reviewed the literature about automatic test case generation. In this survey, various factors for an automate test case generation have been discussed to obtain better efficiency in testing.

Test case automation using optimization approach gives an efficient test suite for the given problem model. Many reviewers have worked on test case generation using Genetic algorithms (Zhang and Wang, 2011; Ahmed and Hermadi, 2008; Mateen *et al*., 2016). Gulia and Chillar (2012) proposed an approach for the generation and optimization of test cases using Genetic algorithms from UML state chart diagrams, on the case study of driverless train. All the possible test paths are generated from the UML state diagram. For the generation of optimized test cases, they have randomly selected few test path sequences and applied Genetic algorithms crossover technique. Efficiency of test cases has been evaluated using mutation analysis. Another approach based on Genetic algorithm, presented by Bala and Chillar (2018) adopts a case study on on-line hospital appointment. Maximum test path have been generated by converting sequence diagram to sequence graph and then optimized very efficiently. Xanthakis *et al*. (1992) proposed a Genetic algorithm which help in generating the test data. As Genetic algorithm are used for generating test data of the structure. User selects the path and all relevant branches executed from programs. By summation of branch predicated fitness function can be calculated. Kaur and Goyal (2011) have presented a Genetic algorithm to prioritize the test cases on the basis of code coverage information. To develop fault detection rate, data on real approaches are used. As a result of this study, TCP method may increase the fault detection rate using clustering approach. A variable length Genetic algorithm has been proposed by Srivastava and Kim (2009) that optimizes and selects path testing average criteria. Genetic algorithm is used in control flow graph. All possible paths are covered in testing in software under test and weights are allocated to the edges of control flow graph by using 80-20 rules. Bala and Chillar (2018) proposed an improved variation of Genetic algorithm which employs mendel operator for prioritization of test cases. The proposed method has been applied on telemedicine simulator such that the testing effort reduces significantly while the code coverage remains almost the same. This is achieved by a novel mendelian operator based Genetic algorithm by following two principles.

Harman (2007) focused on search based software engineering for automated test data generating. Here, Genetic algorithm is used for automated test data generation rely on search based software engineering. Zhang and Wang (2011) use simulated annealing algorithm into Genetic algorithm to generate the test data for path testing. A simulated annealing algorithm is inspired by the annealing of metals. The adaptive Genetic simulated annealing algorithm is proposed by Zhang and Wang (2011) to automatically generate the test data. The steps if this algorithm is shown in his study. The fitness value, crossover, mutation another modification are applied in Genetic algorithm procedure. The majority of software test-data generation techniques are based on Genetic algorithm.

Problem formulation: A telemedicine simulator is considered for optimization of test cases. Telemedicine software is the platform used by providers to connect with patients and share video and images. It is convenient, accessible for patients with least cost, accessible to consultants from specialists with increased patient engagement and better patient care. Some most popular telemedicine solutions specialties are teleradiology, telepsychiatry teledermatology, teleopthamology, etc. (Fig. 1). We have taken a simulation of the telemedicine software for case study. Through this software, we help the patient in disease diagnosis and reducing the consultation time by providing the appointment with the specialist. The procedural steps for appointment module of the

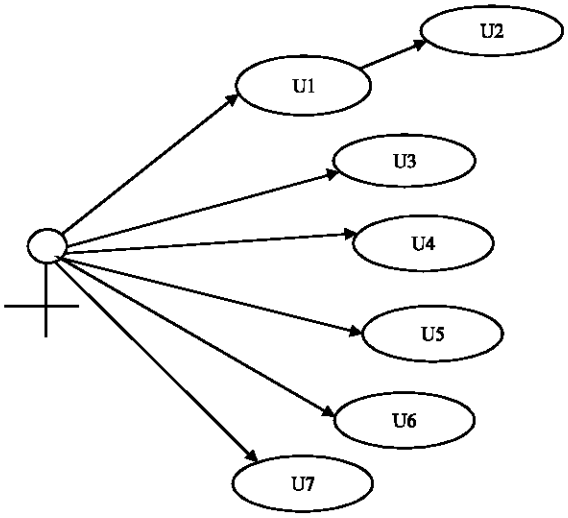

Fig. 1: Use case diagram

multi-agent system for diagnosis are as follows: the patient request for appointment by using the patient login file which invokes the appointment agent. The corresponding agent interface with the patient agent and main agent with its main function being to give confirmed appointment information to the appointment requesters. The main agent interface with appointment agent and doctor agent. The purpose of core in main agent is to cross examine the request of appointment against doctor schedule to provide accessible appointment slots. The purpose of core in the doctor agent is to obtain doctor's schedule distantly and interface with the doctor's appointment database and the schedule agent. The schedule agent interfaces with the main agent and gives proof of doctor's schedule and confirmed appointments.

Following database tables have been used in the proposed scheme: doctor data, patient's data, doctor-expertise, disease-expertise, disease-test, disease-symptoms, doctor-authentication, doctor-patient, patient-test, doctor schedule, patient prescription.

MATERIALS AND METHODS

We here enlighten our research to automatic generation and optimization of test cases using UML artifacts. UML is standard language Unified Modeling Language (UML) was released by Object Management Group (OMG) in 1997 (Mateen *et al*., 2016). It is a modeling language in software engineering. It is being designed to specify, construct and document to software artifacts with support to special aspects of software such as dynamics and structural aspects. Approach used to generate and optimize test cases.

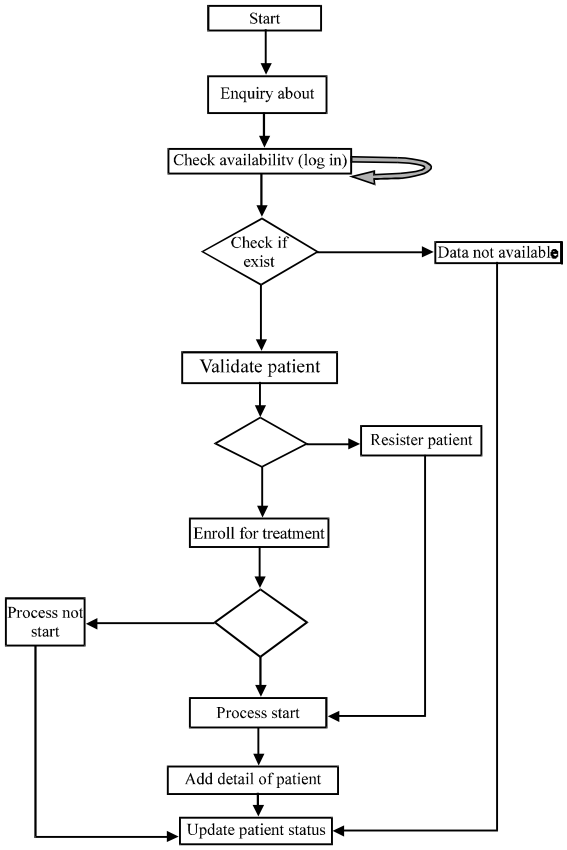

Fig. 2: Activity diagram

- UML diagram
- Use case diagram
- Sequence diagram
- Activity diagram
- Genetic algorithm

Use case diagram: Figure 1 is discussed in use case diagram. U1 = Resisteration; U2 = Fill on line query performa; U3 = Login (using paptient id); U4 = Query for expertise; U5 = Given appointment; U6 = Not given appointment; U7 = Consult to doctor.

Activity diagram: Figure 2 is discussed in activity diagram.

Sequence diagram: Figure 3 is discussed in sequence diagram.

Conversion of sequence diagram to sequence graph: Figure 4 is discussed in conversion of sequence diagram to sequence graph.

Integration of all graphs: Figure 5 is discussed in integration of all graphs.

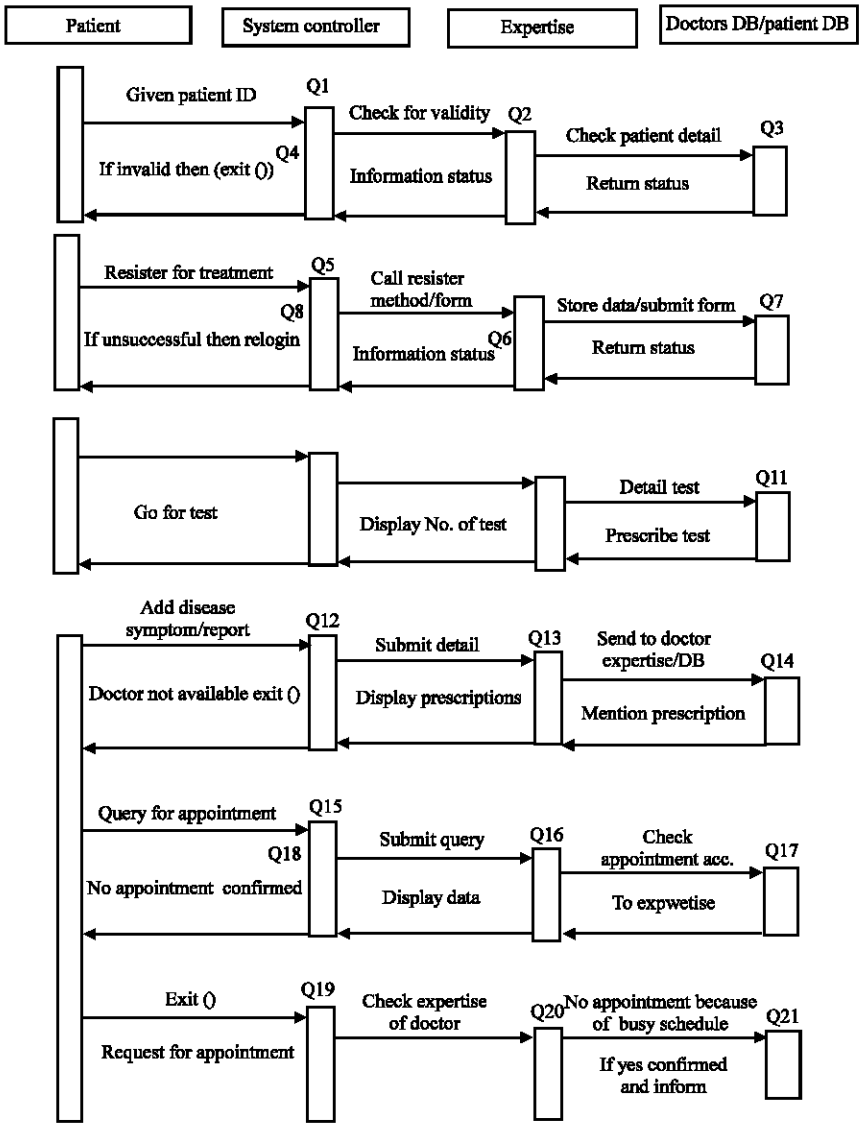

Fig. 3: Sequene diagram

How to calculate weight of total unsuccessful path: 37+38+39+10+14+21+24+28 = 211.

Test path generation using integrated diagram: Unsuccessful end = ue successful end = se. U1-U7 = 1, U7-S4 = 9, S4-unsuccessful = 10, S8-unsuccessful = 14, S8-S15 = 20, S15-unsuccessful = 21, S15-S18 = 23 S18-successful = 28, S21-successful-A11 = 36, S22-unsuccessful = 28, U6-unsuccessful = 37, U3-unsuccesful = 38, U2-unsuccessful = 39, unsuccessful-A4-successful end = 40, PATH 1 = 306, PATH 2 = 358, PATH 3 = 484, PATH 4 = 553, PATH 5 = 659, PATH 6 = 668, PATH 7 = 289, PATH 8 = 290. Figure 6 is discussed in test path generation using integrated diagram.

Algorithm 1; Generation and optimization of test paths:

GEN-Automatic (AUT)-OPT Testcases
Input: -Use Case, Activity Graph (AG), Sequence Graph (SG) == (SUT)
Output: -Optimized test path
1. Discover all the paths, P = {p1, p2, p3, p4, ...,} from start node to the end node in SUT
2. Weights are allocated to path in ascending order from left to right and up to down. Each nodes are being assigned weights and sibling weights are calculated from the parent node
3. Next is to calculate the cost (x) of each path by adding the weights of all succeeding nodes on that path
4. Genetic algorithm is being applied to the SUT
5. Calculate fitness value
   a. Calculate the fitness of cost (x) for each path
   b. Fitness function is calculated by F (x) = x*x
   c. Probability for individual is calculated as $P(I) = F(x)/\sum F(x)$
7. To generate new population best traits are being selected from existing or initial population for mating
Roulette Wheel Selection:

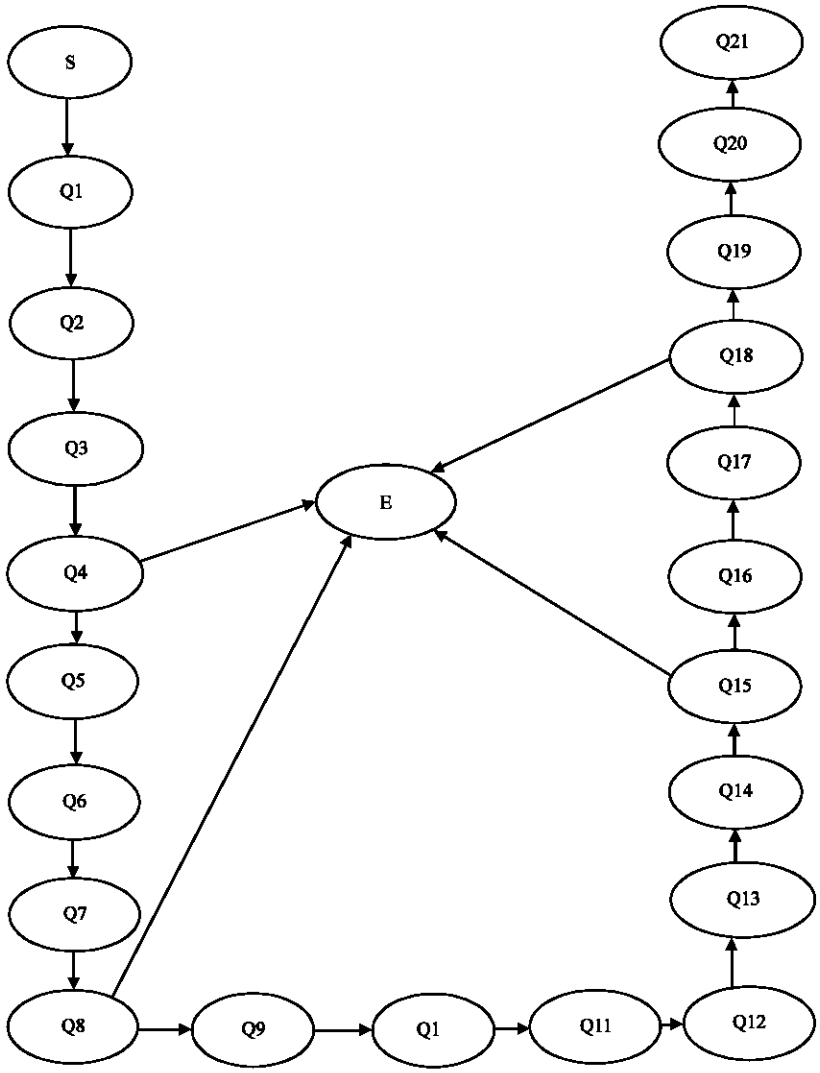

Fig. 4: Conversion of sequence diagram to sequence graph

i. Calculate the fitness value eval (vi) for each chromosome v_i(i = 1,…, pop_size)

ii. Find the total fitness of the population: $\sum_{i=1}^{pop\_size} eval(\upsilon_i)$

iii. Calculate the probability of a selection pi for each chromosomes v_i (I = 1, ..., pop_size)

$$P_i = eval(\upsilon_i)/f$$

iv. Calculate the cumulative probability qi for each chromosome v_i(i=1,…, pop_size)

$$q_i = \sum_{j=1}^{i} p_j$$

The selection process is based on spinning the roulette wheel pop_size times; Each time we select a single chromosome for a new population in the following way:

i. Generate a random number r from the range [0,,,1]

ii. If r<qi then select the first chromosome (v_i); otherwise select the i-th chromosome υi (2≤i≤pop_size) such that $q_{i-1}<r\leq q_i$

8. Crossover: For each chromosome in the (new) population:

i. Generate a random(float) number r from the range [0,,1]

ii. If $r<p_{ci,}$ (0.8) select given chromosome for crossover

iii. Now we mate selected chromosome randomly: for each pair of coupled chromosome we generate a random integer number pos from the range [1, m-1] (m is the total length----number of bits-----in a chromosome). The number pos indicates the position of the crossing point

Two chromosome

$(b_1b_2, ..., b_{pos}\, b_{pos+1}, ..., b_m)$ and

$(c_1c_2, ..., c_{pos}c_{pos+1}, ..., c_m)$

Are replaced by a pair of their offspring:

$(b_1b_2, ..., b_{pos}\, b_{pos+1}, ..., b_m)$ and

$(c_1c_2, ..., c_{pos}c_{pos+1}, ..., c_m)$

9. Mutation: Mutation is performed on bit by bit basis. Another parameter of the genetic system, probability of mutation p_{m1} gives us the expected number of mutated bits p_m.m. pop_size. Every bit(in all chromosome in whole population) has an equal chance to undergo mutation, i.e., change from 0-1 or vice versa. So, we proceed in the following way

For each chromosome in the current, (i.e., after crossover) population and for each bit within the chromosome:

i. Generate a random (float) number r from the range [0,..., 1]

ii. If $r<p_{m\,(0.1),}$ mutate the bit

10. If Feasible:

11. Following selection, crossover and mutation the new population is ready for its next evaluation

This complete process is rehearsed till the minimization of the fitness value

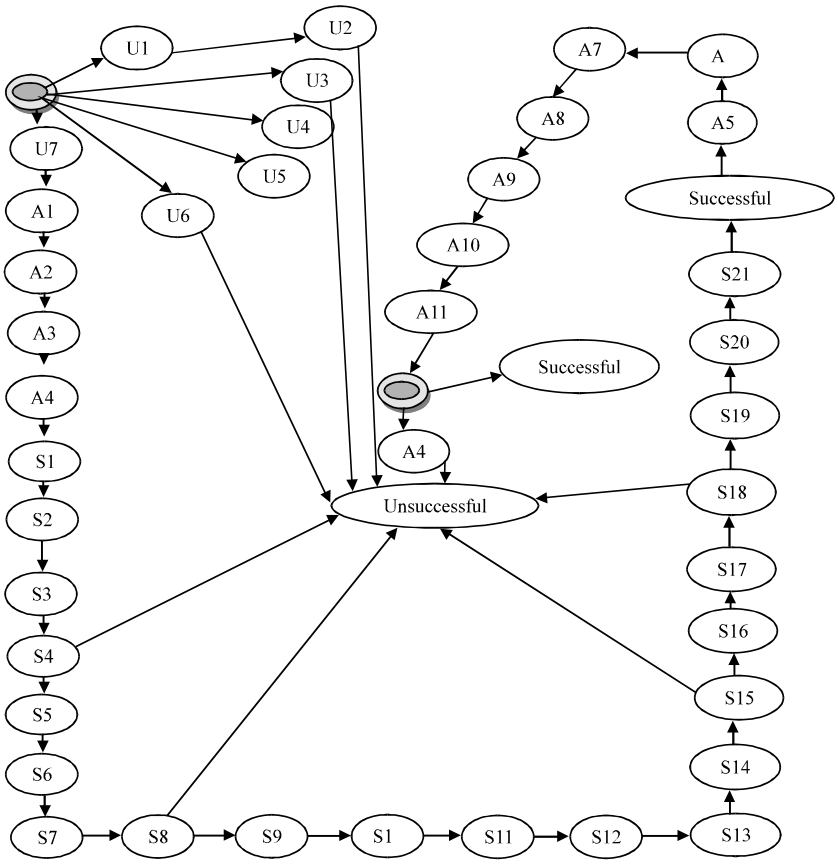

Fig. 5 : Integration of use case, activity diagram, SDG into System Under Test (SUT)

Path 1: u7→A1→A2→A3→A4→S1→S2→S3→S4→UE→A4→SE = 306

Path 2: u7→A1→A2→A3→A4→S1→S2→S3→S4→S5→S6→S7→S8
UE→A4→SE = 358

Path 3: u7→A1→A2→A3→A4→S1→S2→S3→S4→S5→S6→S7→S8
S9→S10→S11→S12→S13→S14→S15→UE→A4→SE = 484

Path 4: u7→A1→A2→A3→A4→S1→S2→S3→S4→S5→S6→S7→S8
S9→S10→S11→S12→S13→S14→S15→S16→S17→S18→UE→A4 SE = 553

Path 5: u7→A1→A2→A3→A4→S1→S2→S3→S4→S5→S6→S7→S8
S9→S10→S11→S12→S13→S14→S15→S16→S17→S18→S19→S20
S21→S22→UE→A4→SE = 659

Path 6: u7→A1→A2→A3→A4→S1→S2→S3→S4→S5→S6→S7→S8
S9→S10→S11→S12→S13→S14→S15→S16→S17→S18→S19→S20
S21→S22→Successful→A5→A6→A7→A8→A9→A10→A11→SE = 668

Path 7: U3→UE→A4→SE = 289

Path 8: U2→UE→A4→SE = 289

Fig. 6: Test path generation using integrated

or the maximum number of generations is reached or all the scenarios have been traversed

12. Test cases are optimized by generating the best scenario as output
13. End

RESULTS AND DISCUSSION

Experimental analysis: We have performed experimentation in MATLAB framework. A simulated telemedicine software project has been developed for generating the test cases and detailed description of Genetic algorithm is given here. All the results are generated using the following parameters of the Genetic algorithms as listed in Table 1. Parameters have been tuned after some experimentation (Table 1-25).

The possible solutions generated from GA are given in Table 1-4. We have recorded the solutions after every

Table 1: Empirical parameters for GA

Variables	Values
Max No. of generations	40
Crossover Probability (Pc)	0.8
Mutation Probability (Pm)	0.1
Population	8
Selection operator	Roulette wheel
Crossover operator	Single point
Mutation operator	Bit flip
Encoding method	Binary

Table 2: Initial population

X	x*x	Chromosomes										P(i)	C(i)	Random No.
668	446224	1	0	1	0	0	1	1	1	0	0	0.2465	0.2465	0.2369
659	434281	1	0	1	0	0	1	0	0	1	1	0.2399	0.4864	0.2289
553	305809	1	0	0	0	1	0	1	0	0	1	0.1689	0.6554	0.6489
484	234256	0	1	1	1	1	0	0	1	0	0	0.1294	0.7848	0.6397
358	128164	0	1	0	1	1	0	0	1	1	0	0.0708	0.8556	0.5896
306	93636	0	1	0	0	1	1	0	0	1	0	0.0517	0.9073	0.6896
290	84100	0	1	0	0	1	0	0	0	1	0	0.0464	0.9538	0.7986
289	83521	0	1	0	0	1	0	0	0	0	1	0.0461	1	0.9250

Table 3: Population after roulette wheel selection

X	Chromosomes										Random No.	For crossover (Y/N)
668	1	0	1	0	0	1	1	1	0	0	0.8669	N
668	1	0	1	0	0	1	1	1	0	0	0.0862	Y
553	1	0	0	0	1	0	1	0	0	1	0.3664	Y
553	1	0	0	0	1	0	1	0	0	1	0.3691	Y
553	1	0	0	0	1	0	1	0	0	1	0.6850	N
484	0	1	1	1	1	0	0	1	0	0	0.5979	Y
358	0	1	0	1	1	0	0	1	1	0	0.7893	N
289	0	1	0	0	1	0	0	0	0	1	0.3676	Y

Table 4: Chromosomes after crossover

X	Chromosomes									
668	1	0	1	0	0	1	1	1	0	0
552	1	0	0	0	1	0	1	0	0	0
485	0	1	1	1	1	0	0	1	0	1
669	1	0	1	0	0	1	1	1	0	1
553	1	0	0	0	1	0	1	0	0	1
552	1	0	0	0	1	0	1	0	0	0
358	0	1	0	1	1	0	0	1	1	0
289	0	1	0	0	1	0	0	0	0	1

Table 5: Mutation site

Row No.	Column No.
7	6
3	9
4	3
5	9
3	8
6	8

Table 6: Chromosomes after mutation

X	Chromosomes after mutation									
668	1	0	1	0	0	1	1	1	0	0
552	1	0	0	0	1	0	1	0	0	0
483	0	1	1	1	1	0	0	0	1	1
541	1	0	0	0	0	1	1	1	0	1
555	1	0	0	0	1	0	1	0	1	1
556	1	0	0	0	1	0	1	1	0	0
374	0	1	0	1	1	1	0	1	1	0
289	0	1	0	0	1	0	0	0	0	1

Table 7: Feasible chromosomes

X	Feasible chromosomes									
668	1	0	1	0	0	1	1	1	0	0
553	1	0	0	0	1	0	1	0	0	1
484	0	1	1	1	1	0	0	1	0	0
553	1	0	0	0	1	0	1	0	0	1
553	1	0	0	0	1	0	1	0	0	1
553	1	0	0	0	1	0	1	0	0	1
358	0	1	0	1	1	0	0	1	1	0
289	0	1	0	0	1	0	0	0	0	1

Table 8: Population after 10 generations

X	x*x	Chromosomes									P(i)	C(i)	Random No.
668	446224	1	0	1	0	0	1	1	1	0	0	0.2465	0.2465	0.2356
659	434281	1	0	1	0	0	1	0	0	1	1	0.2399	0.4864	0.2256
553	305809	1	0	0	0	1	0	1	0	0	1	0.1689	0.6554	0.6256
484	234256	0	1	1	1	1	0	0	1	0	0	0.1294	0.7848	0.5689
358	128164	0	1	0	1	1	0	0	1	1	0	0.0708	0.8556	0.4589
306	93636	0	1	0	0	1	1	0	0	1	0	0.0517	0.9073	0.7895
290	84100	0	1	0	0	1	0	0	0	1	0	0.0464	0.9538	0.8695
289	83521	0	1	0	0	1	0	0	0	0	1	0.0461	1	0.9568

Table 9: After selection

X	Chromosomes										Random No.	For crossover (Y/N)
668	1	0	1	0	0	1	1	1	0	0	0.5974	Y
668	1	0	1	0	0	1	1	1	0	0	0.3353	Y
553	1	0	0	0	1	0	1	0	0	1	0.2992	Y
553	1	0	0	0	1	0	1	0	0	1	0.4525	Y
553	1	0	0	0	1	0	1	0	0	1	0.4226	Y
358	0	1	0	1	1	0	0	1	1	0	0.3596	Y
289	0	1	0	0	1	0	0	0	0	1	0.5583	Y
289	0	1	0	0	1	0	0	0	0	1	0.7425	N

Table 10: After crossover

X	Chromosomes after crossover									
556	1	0	0	0	1	0	1	1	0	0
556	1	0	0	0	1	0	1	1	0	0
361	0	1	0	1	1	0	1	0	0	1
665	1	0	1	0	0	1	1	0	0	1
665	1	0	1	0	0	1	1	0	0	1
550	1	0	0	0	1	0	0	1	1	0
289	0	1	0	0	1	0	0	0	0	1
289	0	1	0	0	1	0	0	0	0	1

Table 11: Mutation site

Row No.	Column No.
2	1
4	1
4	6
6	8

Table 12: Chromosomes after mutation

X	Chromosomes after mutation									
556	1	0	0	0	1	0	1	1	0	0
44	0	0	0	0	1	0	1	1	0	0
361	0	1	0	1	1	0	1	0	0	1
137	0	0	1	0	0	0	1	0	0	1
665	1	0	1	0	0	1	1	0	0	1
546	1	0	0	0	1	0	0	0	1	0
289	0	1	0	0	1	0	0	0	0	1
289	0	1	0	0	1	0	0	0	0	1

Table 13: Chromosomes after feasible

X	Chromosomes after feasible									
553	1	0	0	0	1	0	1	0	0	1
289	0	1	0	0	1	0	0	0	0	1
358	0	1	0	1	1	0	0	1	1	0
289	0	1	0	0	1	0	0	0	0	1

Table 13: Continue

X	Chromosomes after feasible									
668	1	0	1	0	0	1	1	1	0	0
553	1	0	0	0	1	0	1	0	0	1
289	0	1	0	0	1	0	0	0	0	1
289	0	1	0	0	1	0	0	0	0	1

Table 14: Population after 20 generations

X	X*x	Chromosomes										P(i)	C(i)	Random No.
668	446224	1	0	1	0	0	1	1	1	0	0	0.2465	0.2465	0.1256
659	434281	1	0	1	0	0	1	0	0	1	1	0.2399	0.4864	02145
553	305809	1	0	0	0	1	0	1	0	0	1	0.1689	0.6554	0.5689
484	234256	0	1	1	1	1	0	0	1	0	0	0.1294	0.7848	0.7895
358	128164	0	1	0	1	1	0	0	1	1	0	0.0708	0.8556	0.6598
306	93636	0	1	0	0	1	1	0	0	1	0	0.0517	0.9073	0.8659
290	84100	0	1	0	0	1	0	0	0	1	0	0.0464	0.9538	0.8659
289	83521	0	1	0	0	1	0	0	0	0	1	0.04614	1	0.9856

Table 15: Chromosomes after selection

Chromosomes after selection										Random No.	Chromosomes No. XXX for crossover (Y/N)
668	1	0	1	0	0	1	1	1	0	0	0.652451	N
668	1	0	1	0	0	1	1	1	0	0	0.604991	N
659	1	0	1	0	0	1	0	0	1	1	0.387245	Y
358	0	1	0	1	1	0	0	1	1	0	0.142187	Y
358	0	1	0	1	1	0	0	1	1	0	0.025135	Y
306	0	1	0	0	1	1	0	0	1	0	0.421112	Y
290	0	1	0	0	1	0	0	0	1	0	0.618410	N
289	0	1	0	0	1	0	0	0	0	1	0.725775	N

Table 16: Chromosomes after crossover

X	Chromosomes after crossover									
668	1	0	1	0	0	1	1	1	0	0
668	1	0	1	0	0	1	1	1	0	0
355	0	1	0	1	1	0	0	0	1	1
310	0	1	0	0	1	1	0	1	1	0
662	1	0	1	0	0	1	0	1	1	0
354	0	1	0	1	1	0	0	0	1	0
290	0	1	0	0	1	0	0	0	1	0
289	0	1	0	0	1	0	0	0	0	1

Table 17: Population after 30 generations

X	X*x	Chromosomes										P(i)	C(i)	Random No.
668	446224	1	0	1	0	0	1	1	1	0	0	0.246534	0.246534	0.18590
659	434281	1	0	1	0	0	1	0	0	1	1	0.239935	0.486469	0.23560
553	305809	1	0	0	0	1	0	1	0	0	1	0.168956	0.655425	0.20180
484	234256	0	1	1	1	1	0	0	1	0	0	0.129424	0.784849	0.17890
358	128164	0	1	0	1	1	0	0	1	1	0	0.070809	0.855658	0.35680
306	93636	0	1	0	0	1	1	0	0	1	0	0.051733	0.907391	0.78560
290	84100	0	1	0	0	1	0	0	0	1	0	0.046464	0.953856	0.85460
289	83521	0	1	0	0	1	0	0	0	0	1	0.046144	1	0.86540

Table 18: Chromosomes after mutation

X	Chromosomes after mutation									
668	1	0	1	0	0	1	1	1	0	0
668	1	0	1	0	0	1	1	1	0	0
867	1	1	0	1	1	0	0	0	1	1
310	0	1	0	0	1	1	0	1	1	0
658	1	0	1	0	0	1	0	0	1	0
354	0	1	0	1	1	0	0	0	1	0
802	1	1	0	0	1	0	0	0	1	0
289	0	1	0	0	1	0	0	0	0	1

Table 19: Chromosomes After feasible

X	Chromosomes after feasible									
668	1	0	1	0	0	1	1	1	0	0
668	1	0	1	0	0	1	1	1	0	0
668	1	0	1	0	0	1	1	1	0	0

Table 19: Continue

X	Chromosomes after feasible									
306	0	1	0	0	1	1	0	0	1	0
659	1	0	1	0	0	1	0	0	1	1
358	0	1	0	1	1	0	0	1	1	0
668	1	0	1	0	0	1	1	1	0	0
289	0	1	0	0	1	0	0	0	0	1

Table 20: Mutation site

Row No.	Column No.
7	1
5	8
3	1

Table 21: Chromosomes after selection

	Chromosomes after selection										Random No.	Chromosomes No. XXX for crossover (Y/N)
668	1	0	1	0	0	1	1	1	0	0	0.766831	N
668	1	0	1	0	0	1	1	1	0	0	0.336699	Y
668	1	0	1	0	0	1	1	1	0	0	0.662382	N
668	1	0	1	0	0	1	1	1	0	0	0.244165	Y
659	1	0	1	0	0	1	0	0	1	1	0.295507	Y
358	0	1	0	1	1	0	0	1	1	0	0.680178	N
306	0	1	0	0	1	1	0	0	1	0	0.527847	Y
289	0	1	0	0	1	0	0	0	0	1	0.411594	Y

Table 22:

X	Chromosomes after crossover									
668	1	0	1	0	0	1	1	1	0	0
668	1	0	1	0	0	1	1	1	0	0
668	1	0	1	0	0	1	1	1	0	0
284	0	1	0	0	0	1	1	1	0	0
659	1	0	1	0	0	1	0	0	1	1
358	0	1	0	1	1	0	0	1	1	0
690	1	0	1	0	1	1	0	0	1	0
289	0	1	0	0	1	0	0	0	0	1

Tabel 23: Mutation site

Row No.	Column No.
6	5

Table 24: Chromosomes after mutation

X	Chromosomes after mutation									
668	1	0	1	0	0	1	1	1	0	0
668	1	0	1	0	0	1	1	1	0	0
668	1	0	1	0	0	1	1	1	0	0
284	0	1	0	0	0	1	1	1	0	0
659	1	0	1	0	0	1	0	0	1	1
326	0	1	0	1	0	0	0	1	1	0
690	1	0	1	0	1	1	0	0	1	0
289	0	1	0	0	1	0	0	0	0	1

Table 25: Feasible chromosomes

X	Feasible chromosomes									
668	1	0	1	0	0	1	1	1	0	0
668	1	0	1	0	0	1	1	1	0	0
668	1	0	1	0	0	1	1	1	0	0
289	0	1	0	0	1	0	0	0	0	1
659	1	0	1	0	0	1	0	0	1	1
306	0	1	0	0	1	1	0	0	1	0
668	1	0	1	0	0	1	1	1	0	0
289	0	1	0	0	1	0	0	0	0	1

10 generations (Table 1). Represents the initial population which is randomly generated. Probabilities and cumulative probabilities for selection are computed for Roulette Wheel selection. After applying Roulette Wheel selection next table represents the selected solutions. The subsequent tables contain the solutions after crossover

and mutation. Then the solutions obtained are converted into feasible solutions. This is done by calculating the Euclidean distance between the infeasible solution and two nearest solutions of that solution. The infeasible solution is mapped into the solution having least Euclidean distance.

CONCLUSION

Activity graph and sequence graph show test paths which are being optimized using Genetic algorithm. Genetic algorithm is heuristic based search method by exploring good multidimensional search by maintaining an optimized population, random actions consisting of the combination regarding iterative search steps.

REFERENCES

Ahmed, M.A. and I. Hermadi, 2008. GA-based multiple paths test data generator. Comput. Oper. Res., 35: 3107-3124.

Bala, A. and R.S. Chhillar, 2018. A novel test case prioritization using mendel operator based genetic algorithm. J. Theor. Appl. Inf. Technol., 96: 2567-2577.

Gulia, P. and R.S. Chillar, 2012. A new approach to generate and optimize test cases for UML state diagram using Genetic algorithm. ACM. SIGSOFT. Software Eng. Notes, 37: 1-5.

Harman, M., 2007. Automated test data generation using search based software engineering. Proceedings of the 2nd International Workshop on Automation of Software Test (AST '07), May 20-26, 2007, IEEE, Minneapolis,Minnesota,ISBN:978-0-7695-2971-2,pp: 2-2.

Kaur, A. and S. Goyal, 2011. A genetic algorithm for regression test case prioritization using code coverage. Int. J. Comput. Sci. Eng., 3: 1839-1847.

Maheshwari, V. and M. Prasanna, 2015. Generation of test case using automation in software systems-a review. Indian J. Sci. Technol., 8: 1-9.

Mateen, A., M. Nazir and S.A. Awan, 2016. Optimization of test case generation using Genetic Algorithm (GA). Intl. J. Comput. Appl., 151: 6-14.

Sabharwal, S., R. Sibal and C. Sharma, 2010. Prioritization of test case scenarios derived from activity diagram using Genetic algorithm. Proceedings of the 2010 International Conference on Computer and Communication Technology (ICCCT), September 17-19, 2010, IEEE, Allahabad, India, ISBN:978-1-4244-9033-2, pp: 481-485.

Srivastava, P.R. and T.H. Kim, 2009. Application of Genetic algorithm in software testing. Int. J. Software Eng. Appl., 3: 87-96.

Sumalatha, V.M. and G.S.V.P. Raju, 2013. Object oriented test case generation technique using Genetic algorithms. Intl. J. Comput. Appl., 61: 20-26.

Xanthakis, S., C. Ellis, C. Skourlas, A.L. Gall and S. Katsikas *et al.*, 1992. Application of Genetic algorithms to software testing. Proceedings of the 5th International Conference on Software Engineering and Applications, December 7-11, 1992, University of Toulouse, Toulouse, France, pp: 625-636.

Zhang, B. and C. Wang, 2011. Automatic generation of test data for path testing by Adaptive Genetic Simulated Annealing algorithm. Proceedings of the 2011 IEEE International Conference on Computer Science and Automation Engineering (CSAE), June 10-12, 2011, IEEE, Shanghai, China, ISBN:978-1-4244-8727-1, pp: 38-42.