

Evolution or revolution: the critical need in genetic algorithm based testing

Anupama Surendran¹ · Philip Samuel²

Published online: 27 August 2016
© Springer Science+Business Media Dordrecht 2016

Abstract Software testing is one of the most inevitable processes in software development. The field of software testing has seen an extensive use of search based techniques in the last decade. Among the search based techniques, it is the metaheuristic techniques such as genetic algorithm that has garnered the major share of attention from researchers. Looking at the large body of work that has happened and is happening in this field, we feel that it is high time someone studied how well genetic algorithm based techniques fare in practical testing process. Method: In this work, we present a roadmap to the future of genetic algorithm based software testing, based on a review of literature. We have mainly reviewed the works which use genetic algorithm for software test data generation. This independent review is designed to direct the attention of future researchers to the deficiencies of genetic algorithm based testing, their possible solutions and the extent to which they are correctable. The observations from the selected primary studies highlight the issues faced when genetic algorithm is applied in software testing. The observations from the review reveal that the type of genetic algorithm used, fitness function design, population initialization and parameter settings does impact the quality of solution obtained in software testing using genetic algorithm. From the review we conclude that, more generalized approaches can make genetic algorithm based software testing one of the strongest methods in practical software testing. We hope that, this review will be a major breakthrough in genetic algorithm based software testing field.

Keywords Software testing · Genetic algorithms · Review · Population · Parameter settings · Selection · Crossover · Mutation · Fitness function design

✉ Anupama Surendran
anudeepaknair@gmail.com

¹ Department of Computer Science, Cochin University of Science & Technology, Kochi, Kerala, India

² Information Technology Division, SOE, Cochin University of Science & Technology, Kochi, Kerala, India

1 Introduction

Software has an irreplaceable role in our day to day life and this makes software testing a vital cog in the wheel (Korel 1990; Myers 1979). Even though software testing has evolved tremendously, it still remains both time consuming and personnel dependent. Almost 50 % of the cost and time of the software development is taken up by testing (Beizer 1990; Jorgensen 2008). Hence, automation of software testing assumes paramount importance. Among automated testing process, genetic algorithm (GA) based testing has gained significant momentum from 2000 onwards (McMinn 2005, 2011). In spite of the large number of works in the field of genetic algorithm based software testing, the real issues faced by testers are yet to be addressed. Our work tries to resolve this problem by reviewing genetic algorithm based testing works.

1.1 Software testing

Software testing is one of the most important phases in software development (Beizer 1990). To a great extent, automating the process of software testing could reduce the total cost, time and effort incurred during software testing. Black-box testing and white-box testing are the main two types of testing. In Black box testing, the source code of the program is not considered. Program functionality is only considered in black-box testing. In white-box testing the source code of the program is considered. It is also known as structural testing. In structural testing, the fault is revealed by executing the program with selected input and by evaluating the correctness of the program behaviour against the expected ones. Generation of test data is one of the most important steps in structural testing. A test data consists of an input, a set of statements on which the data is applied and the corresponding output. A test case is a collection of test data (Jorgensen 2008). In order to find the test data, the program should be executed with all combinations of input from program input domain. This is one of the most difficult problems faced by the testers. In this review, we have mostly considered the works which use genetic algorithm for test data and test case generation during structural testing.

1.2 Test data generation methods

This section gives an overview of various test data generation methods used for structural testing. Test data generation is the process of finding program input on which a particular program statement (branch statement, condition statement or program path) is executed. Random search is one of the basic and simplest methods of finding test data (Michael et al. 2001). The program statements are executed using randomly generated input values. Though this is one of the simplest methods, random search cannot guarantee success every time. Therefore, more guaranteed approaches like path oriented approach are considered. In this approach, the problem of test data generation is considered as a ‘path’ problem. The path for which test data is to be generated is selected automatically (Harman and McMinn 2010). This path in turn leads to the destination. If the selected path doesn’t lead to the target path, then another path is considered. This process is continued until the target path is found out or until the required test data is found or until the time specified for data generation is over. Symbolic execution and execution-oriented test data generation is used to generate test data in path oriented approach. In symbolic execution, there will be a set of constraints and these constraints must be satisfied for traversing the paths in path-oriented approach. Symbolic execution faces difficulties when the program is very complex and long. In execution oriented

approach, the actual execution of the program occurs. This is a goal oriented approach where the process of finding input is represented as a set of sub goals. Here the program is executed with the randomly generated input. For the generated input, the program execution flow is traced. As the program execution progresses, the search procedure decides whether the execution should proceed or whether an alternate branch is to be considered as the current path may not lead to the goal. If an undesirable execution at a branch occurs, then a real valued function is associated with the branch. Function minimization search algorithm is used to find new input, which may alter program execution flow at the current branch. This process is continued until the target node or goal is obtained. In path oriented approach, the main problem occurs when non-executable paths are explored which causes a loss in computation effort (Harman et al. 2012).

Another method used for test data generation is the chaining approach. This method may be considered as an extension of path oriented approach. In chaining approach, dataflow analysis is used for generating test data. The program statements are represented as nodes. Edges represent the interconnection between the nodes. An input value is randomly generated and the program is executed with this input. If the execution of the program with the generated input leads to a branch which does not lead to the target node, then a real valued function is associated with this node. Function minimization search algorithm is applied to find new input value. The new input alters the program execution flow at the current node. Here the execution flow is altered in order to find suitable input value. Chaining method also faces difficulties as the program size and complexity increases. Metaheuristic search techniques offer great support in such situations and this may be used for structural, functional and non-functional testing (Harman 2007; Shamshiri et al. 2015). In metaheuristic techniques, the test data generation problem can be represented amenable to symbolic execution. A fitness function is designed for this purpose and a set of manipulations and operators are applied to optimize this fitness function. Hill-climbing, Simulated annealing, Genetic algorithms, Swarm-Intelligence techniques are some of the commonly used metaheuristic search techniques for test data generation.

Among the metaheuristic search techniques, Hill-climbing is one of the simple methods. In hill-climbing, an initial solution is chosen randomly (Harman et al. 2007). Then the current solution is compared with the neighbours. The new solution is considered if the new solution is better than the current. Main disadvantage of this method is that the solution may get stuck in local maxima. Therefore, hill-climbing is considered as a local search method. Simulated annealing gives better result compared to hill-climbing. Single space exploration is used in simulated annealing. Therefore, it is considered as a local search method. In simulated annealing also, the solution may get stuck in local maxima. Genetic algorithm is an adaptive global search method based on the principle of evolution. In genetic algorithm, the chances of getting stuck in local maxima are less compared to hill-climbing and simulated annealing. Particle swarm optimization (PSO) is a global search method. Search space exploration is more efficient in genetic algorithm and particle swarm optimization compared to simulated annealing. The main advantage of using PSO compared to genetic algorithm is that PSO is easier to implement and have few parameters to adjust compared to GA. Compared to GA, even though PSO can quickly find the most appropriate area in the solution for a problem, PSO faces difficulty to find best solution.

1.3 Genetic algorithm in software testing

Genetic algorithm can be used as an effective tool in software testing (Michael et al. 2001; Alshahwan and Harman 2014). For example, consider the testing of a large and complex

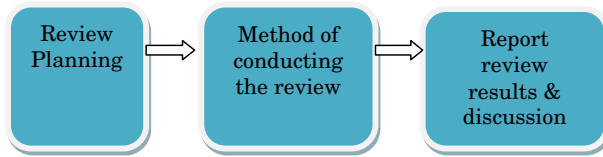


Fig. 1 Stages in the review

program using GA. The tester will have to handle large and complex program code which should satisfy the test adequacy criterion. The adequacy criteria may be branch, statement, path, control flow or data flow coverage. The infinite number of paths in such programs is supposed to satisfy any of these criteria. When the numbers of paths are very large, it may not be possible to generate and execute all combinations of input data from program input domain for satisfying any of the above mentioned coverage criteria. This is one of the major hurdles faced by testers. GA may be used in such situations for generating test data. In such complex problems, where an optimal solution is difficult to achieve, GA may be used to get a good solution where time and system resources are limited. Solution sought by GA is test data which causes the execution of statement, branch or path in structural testing.

2 Systematic study: Why and How?

We have made an attempt to look into the issues of GA based software testing due to many reasons. This section explains the purpose of our study and the steps we have followed to conduct the study. Our review consists of mainly three stages. A diagrammatic representation of the basic stages in our review is given below in Fig. 1 and a detailed representation of the steps in our review is given in Fig. 2.

In the review planning stage, the need for systematic review is highlighted. A review protocol is also developed in the planning stage. Research question identification, selection criteria for works, inclusion and exclusion criteria, quality assessment criteria and data extraction steps are carried out in the review conducting stage. Results of the review work collected are discussed in review report stage.

2.1 Review planning

This is the initial stage in our review. This stage consists of two steps given below

- Highlighting the need for review.
- Developing the review protocol.

2.1.1 Advantages of the review

In the past few years search based software testing, especially evolutionary algorithm, has gained immense popularity (Ahmed and Hermadi 2008; Chen et al. 2009; Harman et al. 2012; McMinn 2011; Ali et al. 2009). A graph is shown in Fig. 3, which shows an increase in the rate of publications and research works in search based software testing during the period 1975–2015 (McMinn 2011; Harman et al. 2015). Among evolutionary algorithms, genetic algorithm based software testing has received a wide interest from researchers due to its ability to handle complex problems where an exact solution does not exist (McMinn 2011).

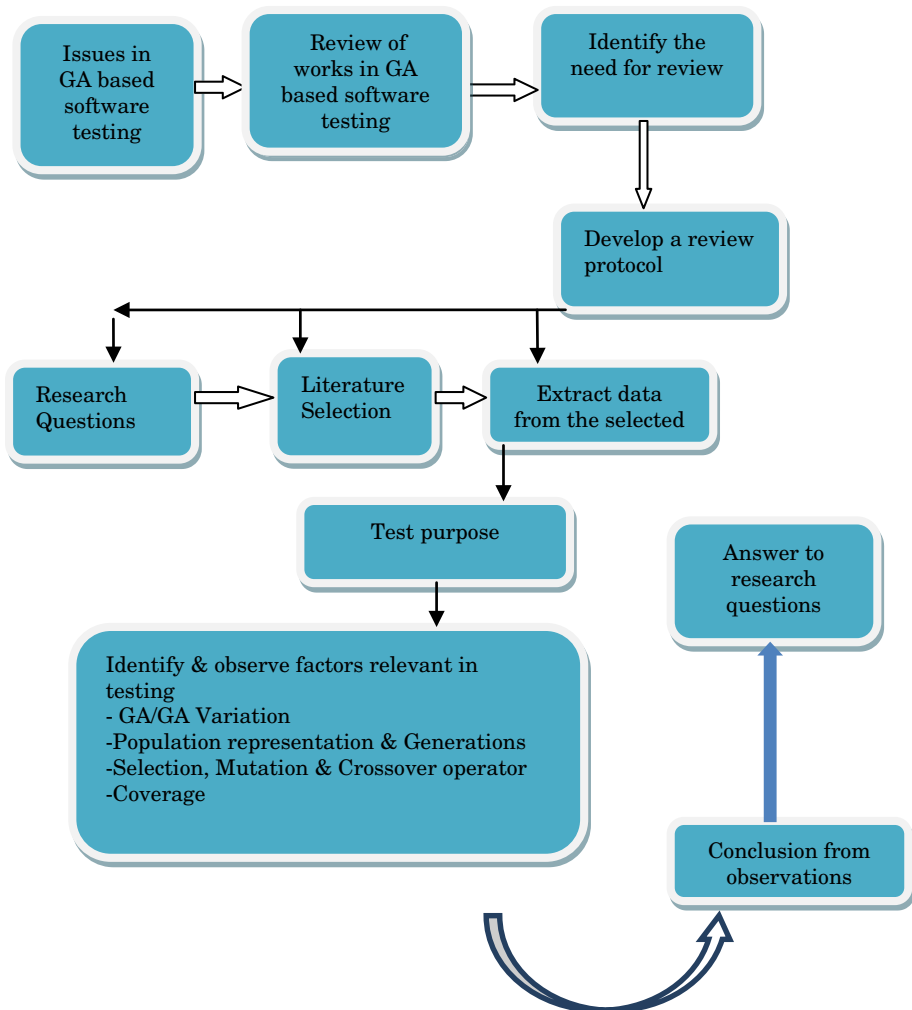


Fig. 2 Detailed steps in the review

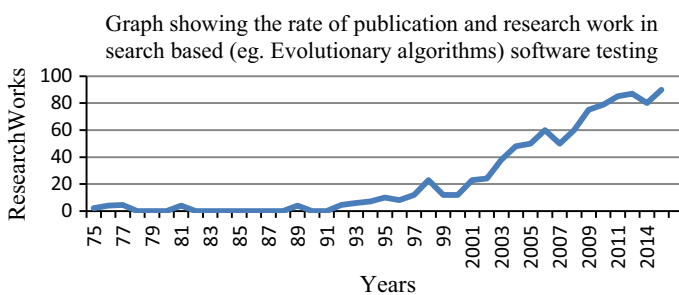


Fig. 3 Research works in search based software testing during the period 1975–2015

The need for systematic review of genetic algorithm based software testing arises due to a number of reasons. Even though genetic algorithm based testing has made a great impact in academic research, only very little attention has been given to understand the complexities of using genetic algorithms in practical software testing.

In this study, we have tried to focus on this problem and have tried to highlight the challenges involved in genetic algorithm based approaches for using it as a practical tool in software testing. The main reason for choosing this problem in our work is because of the usage of genetic algorithms in software testing without knowing the ambiguities in genetic algorithm based testing. In this paper, we have mentioned some works which utilize genetic algorithm for testing. We can see that none of these works have adopted any general operator setting for testing purpose. This inherent non-deterministic nature of the genetic operators makes program testing a demanding task (Feldt and Poulding 2015). The strength of using genetic algorithm mainly depends on setting the genetic parameters to their appropriate values which in turn depends on the problem to be solved (Binkley et al. 2009; Xanthakis et al. 1992). This itself is a major challenge faced by testers. We have mentioned some of these challenges and have also pointed out the factors that are still not considered by researchers during GA based software testing. Making an unbiased review like this may help to solve the issues in genetic algorithm based software testing and at the same time help the future researchers to explore the untouched research areas in GA based software testing.

2.1.2 Review protocol

We have developed a review protocol to effectively cover all the works which comes under our area of interest. Clearly defining a review protocol is inevitable for any systematic study as it forms the basis of the study and helps to carry out the review in an unbiased manner (Kitchenham 2007). All the observations, results and the effectiveness of the study depend on the protocol defined. Our review protocol consists of the following steps given below:

- Identification of research questions
- Selection of works/Source selection
- Inclusion and Exclusion criteria for works
- Quality assessment of the works
- Data extraction and organization

Based on the steps in our review protocol, the review is carried out.

2.2 Conducting the review

Based on the review protocol defined in the first stage, the review is carried out. Different steps present in conducting the review are explained in detail in the forthcoming sections. Identification of research questions is the first step in conducting the review. We have provided a brief explanation of genetic algorithm before going through the various steps in conducting the review.

2.2.1 Brief overview of Genetic algorithm

This section gives an overview of the basic steps involved in genetic algorithm. Genetic algorithm is a type of evolutionary algorithm and is considered as one of the best of all evolutionary algorithms (Goldberg 1989). It is a type of meta heuristic search technique developed by John Holland and works on Darwin's principle of survival of the fittest (Holland

Fig. 4 Basic steps of genetic algorithm

```

procedure Genetic Algorithm
begin
  GET (Initial Population);
  CALCULATE FITNESS
  (Initial Population)
loop
  FINALIZE POPULATION
  FOR CROSSOVER
  (Parent population)
  PERFORM CROSSOVER
  (Parent population, child)
  APPLY MUTATION (Child)
  CALCULATE FITNESS (Child)
  GET NEXT GENERATION
  (Parent population, Child)
stop process when
  TERMINATION CRITERIA
exit loop
end

```

1975). Genetic algorithm uses the technique of natural genetics, representing a computer model of biological evolution. Genetic algorithms have the ability to solve a variety of optimization and search problems. Several testing techniques use genetic algorithms believing that testing may be carried out in a better way using the natural evolutionary process present in them.

Genetic algorithm identifies an optimal solution for a problem by applying natural evolutionary techniques to a group of possible solutions referred to as “population” (Girgis 2005; Srinivas 1994; Tracey 2000). After each generation, a new generation is formed which is better than the previous generation. The series of steps involved in genetic algorithm are population initialization, selection, crossover, mutation and termination (Fraser et al. 2013). A string of digits called chromosomes are present and each individual of the string is called a gene. Each individual in the population has a fitness value which decides the quality and performance of that individual. Greater the fitness value better will be the problem solving capacity of an individual (Michael et al. 2001). Collection of chromosomes makes up a population. The initial population is created randomly and the fitness of the individuals in the population is calculated. This information is used to select the best candidates for forming the next generation parents. After selecting parents of the successive generation, the next step is to combine these candidates to form the offspring. Crossover operation is used to perform this step. Crossover enables the selection of good features from parents to form the offspring. Mutation is applied to the offspring to create better quality individuals. Mutation is defined as the process of altering the genes in the chromosome. A new generation is chosen from the offspring based on the fitness of the individuals. These individuals are considered as parents of the next generation. This cycle is repeated until a global solution for the problem is obtained. The basic steps of genetic algorithm are given in Fig. 4.

2.2.2 Identification of research questions

Research questions may be considered as the foundation of the any review as it defines the objective or aim of the review and the whole review result is based on the answers of these questions. Systematic review must be carried out with an aim to answer these research

Table 1 Research questions

RQ1	In spite of the large volume of works in genetic algorithm based testing, why some works have considered variations of genetic algorithms?
RQ2	What is the effect of population representation and size in software testing?
RQ3	Is there any common method to design fitness function during software testing?
RQ4	What is the general strategy adopted in operator selection and parameter settings during software testing?
RQ5	What is the significance of computing time and convergence in GA based testing?
RQ6	What is the role of coverage in GA based testing?

Table 2 Source repository

IEEE Xplore (Transactions and Conferences)
ACM digital library (Transactions and Conferences)
Springer (Journals and Conferences)
Wiley (Journals)
Scopus
Scholar

questions. During this process, we may come across several factors which may or may not be related to research questions. These factors may be categorized by some assumptions and observations drawn from various literatures. Here we have defined a research question which is given below.

Can GA based software testing evolve as an attractive technique in software testing industry? If so, what are issues to be sorted out in GA based testing?

In order to answer these questions, we have to consider several factors which improve the quality of GA based software testing. Therefore, we have refined this question into six research questions RQ1, RQ2, RQ3, RQ4, RQ5 and RQ6 which are given below in Table 1. These research questions (RQ) help to emphasize the strengths and significance of our review.

Research question RQ1 is expected to find out the works which use variations of GA in test data generation and is aimed at finding whether the variations of GA prove to be better than GA for test data generation. RQ2 is expected to find out the population settings used in software testing and their impact on the final outcome during testing. RQ3 analyses the different methods used for designing the fitness function. RQ4 analyses the operator and parameter setting used for software testing and the impact caused by the same during testing. RQ5 stresses the significance of computing time and convergence and RQ6 highlights the role of coverage in GA based testing.

2.2.3 Source selection

Since our study is based on the observations from several literatures, the first step is to identify the source or database for literature selection. We have considered some premium conference works as well as works published in transactions and some reputed journals for our reference. Various database used for literature selection are given below in Table 2.

Table 3 Keyword used in source selection

Genetic algorithm
AND
Software
AND
(Test Data OR Test Case)
AND
(Generate OR Find OR Get)

After identifying the repository for selecting the works, the next step is to find out the relevant works in the concerned field. Sect. 2.2.4. explains how the works are selected for review.

2.2.4 Inclusion and exclusion criteria

This section explains the steps in literature selection from the source repository. The first step is to identify or finalize the keyword or query which may be applied to the source repository. The search keyword is formed using Boolean 'AND' and 'OR' expression. The terms are combined using 'AND' and 'OR' is used in between the alternative terms. The final keyword used for primary study selection in our review is given below in Table 3.

As a result, we got nearly 300 research papers from these sources. After a close examination of these papers the works were related to the field of hardware testing, testing of circuits were ignored. After the second round of selection, nearly 120 papers were selected. As it is not possible to review all applications of genetic algorithm based software testing, we have ignored some works which illustrates test case prioritization and test case minimization etc. using genetic algorithms. Keeping these factors in mind, we defined a selection criterion which is relevant for understanding the core issues in using GA for software testing as well as to explore the future research perspectives in GA based software testing. In our selection criteria, the papers which used GA or variation of GA or combination of GA with other methods for improving the test data generation process during software testing were identified (Lin et al. 2012). We also selected some works which uses GA for structural testing and the works which studied GA operator and parameter setting effects during test data generation (Arcuri and Fraser 2013). We ignored the works which use GA for other type of testing such as black box, model based testing etc. (Fischer 2012). Some of the inclusion and exclusion criteria used in our review are given below. Finally, after the second round of iteration, we selected nearly 31 papers which clearly fall within our review scope.

Inclusion criteria

- Using GA for software test data generation
- Using GA for software test case generation
- Using GA for Path testing
- GA based structural testing and coverage
- Study effect of GA parameter settings during testing
- Study effect of GA operator settings during testing

Exclusion criteria

- Using GA for hardware testing, circuit testing etc.

Table 4 Quality assessment form

Quality factors	Description	Outcome/result of quality factors
Q1	Is the aim of the work clearly defined?	Yes/no/average
Q2	Whether the method is clearly explained?	Yes/no/average
Q3	Whether the outcome/result of the method clearly explained?	Yes/no/average
Q4	Whether the limitations of the work/method specified?	Yes/no/average
Q5	Does the study contribute to our review?	Yes/no/average

- Using GA for test data augmentation
- Using GA for software test case minimization
- Using GA for software test case prioritization
- Black box/Model based testing using GA.

2.2.5 Quality assessment of the works

Inorder to assess the quality of the selected primary studies, we have made a quality assessment check in this review. For this, we have prepared a quality assessment form. There were several factors in the quality assessment form. Each factor is given a score can have either a value of 0, 1 or 0.5 which corresponds to 'No', 'Yes' or 'Average'. Each primary work was analysed and a score was assigned for each of these factors (Oxman 1994). For all the works included in the primary study, the value got for each of these factors was noted and the total value was calculated. If the total value was below a certain limit, such works were not considered. Quality assessment of the studies was made inorder to make the review unbiased. Our quality assessment form consists of the following factors.

The quality assessment form used in our work is given in Table 4. We have considered five quality factors for each study. All these factors were considered for each of the work and a score was assigned. The works selected after quality assessment was given a unique number as an identifier. The result of the quality assessment is given in Sect. 2.3.

2.2.6 Data extraction and organization

After assessing the quality of the primary studies in our review, the required data was extracted from each work. The extracted data is supposed to contain all relevant information inorder to answer the research question. The entries of our data extraction form are given in Fig. 5.

Extracted data is given in the result section (Sect. 2.3) of this review. Research questions were answered based on the analysis made on the extracted data.

2.3 Result and discussion

This section deals with the result of the review and the review observations from the discussion.

2.3.1 Result

The works selected after the quality assessment check is listed here. From the selected works, data is extracted according to the data extraction criteria. All the values corresponding to the entries in the data extraction form were noted.

Data Extraction form Entries

- Work Identifier (Number)
- Title of the work
- Description of the work
- Reference of the work
- Works using variations of GA for testing
- Population, selection, crossover &mutation operator
- Parameter settings
- Fitness function design
- Coverage criteria
- Future research directions
- Whether the observations from the works address the research question?

Fig. 5 Entries in data extraction form

Works selected after quality assessment check:- We already saw the quality assessment form used in our review in Sect. 2.2.4. The works selected are given below in Table 5 and they are given a unique identifier. The quality assessment score is also given to reveal the transparency in the selection criteria.

We adopted a quality assessment approach for analyzing the papers and for pointing out the shortcoming present in GA based testing approaches. This helps to highlight the untouched and challenging research areas possible in GA based software testing. We have conducted this review with an aim of answering the research questions listed in Sect. 2.2.2. For that, we have extracted the data from works according to the entries in data extraction form.

Variations of genetic algorithm in software testing:- This is one of the entries in data extraction form which may be used for analyzing the effect of GA in software testing. Even when a group of researchers claim that simple GA is best for software testing, we can see that a lot of works have used several variations of GA for software testing. Simple GA mimics the process of natural evolution with basic steps such as population representation, selection, crossover and mutation. We can see that in software testing, either simple genetic algorithm, adaptive GA, parallel GA, multipopulation GA or some type of hybrid methods which use a combination of GA and some other methods may be used (Lin et al. 2012; Zhang et al. 2015). In most of the works it is mentioned that the variations of GA are used to overcome the limitations of simple GA. We have enumerated here certain works using these. Xue-ying et al. used genetic algorithm for reducing the cost associated with test suite reduction (2005). They have introduced a new type of genetic algorithm called GeA which considers cost factor during the calculation of fitness function. They have compared the performance of their algorithm with Greedy algorithm named MgrA and simple genetic algorithm called SGeA. The performance of GeA was found to be better than SGeA and MgrA (2005). Latiu et al. (2012) used evolutionary algorithms for software path testing. In their work, they compared the result of two new approaches based on particle swarm optimization and simulated annealing with genetic algorithms.

They took the parameter values based on some assumptions made from previous work. Finally, they concluded that simulated annealing based approach is better than genetic algorithm and particle swarm optimization based approach. Malburg and Fraser (2011) have used a hybrid method which combines GA based and constrained based test data generation

Table 5 Works selected after the final round

Identifier	Author and year	Quality assessment score	Type of testing	Purpose
1	Xue-ying et al. (2005)	4.5	Structural testing	Reduce cost associated with test suite reduction using GA
2	Xiao and Afzal (2010)	4.5	Structural testing	Bug fixation with GA
3	Latiu et al. (2012)	5	Structural testing	Software path testing using GA
4	Ahmed and Hermadi (2008)	4.5	Structural testing	Test data generation for multipath testing using GA
5	Pocatuil and Ivan (2013)	5	Structural testing	Automated approach for branch testing using GA
6	Roper et al. (1995)	4.5	Structural testing	Used GA for testing C programs for attaining the branch coverage criteria
7	Jones et al. (1996)	4.5	Structural testing	Branch coverage using GA
8	Pargas et al. (1999)	4.5	Structural testing	Path coverage and Branch coverage using TGen tool
9	Michael et al. (2001)	5	Structural testing	Developed a tool called GADGET which uses genetic algorithms for generating test data for branch coverage of C programs
10	Sofokleous and Andreou (2008)	4.5	Structural testing	Test data generation framework using genetic algorithms to provide edge/partition coverage
11	Chen et al. (2009)	5	Structural testing	Test data generation for branch coverage
12	Bernat	4.5	Structural testing	Studied the effect of response time in GA based testing
13	Khor and Grogono (2004)	4.5	Structural testing	Used the concept of GA and formal concept analysis to generate test data for branches
14	Cao et al. (2009).	4.5	Structural testing	Test data generation of a specific path using GA
15	Malburg and Fraser (2011)	5	Structural testing	Introduced a hybrid method which combines GA based and constrained based test data generation approach
16	Zhang et al. (2007)	5	Structural testing	Test data generation of many paths using GA
17	Fraser and Arcuri (2012)	5	Structural Testing	Studied the effect of seeding in test data generation of object oriented programs
18	Arcuri and Fraser (2011)	5	Structural testing	Studied the effect of parameter settings for object oriented programs using Evosuite tool
19	Bueno and Jino (2002)	4.5	Structural testing	Path coverage testing using GA

Table 5 continued

Identifier	Author and year	Quality assessment score	Type of testing	Purpose
20	Wegener et al. (2001)	5	Structural testing	Test data generator for structural testing of real-world embedded software systems using GA
21	Miller et al. (2006)	4.5	Structural testing	Test data generation for Branch coverage using GA
22	McMinn (2013)	5	Structural testing	Studied the impact of cross over on C programs during GA based testing
23	Fraser et al. (2013)	4.5	Structural testing	Test suite generation for object oriented programs which cover multiple goals simultaneously
24	Gonga et al. (2011)	4.5	Structural testing	Test data generation for many paths using GA
25	Pocatuilo and Ivan (2013)	5	Structural testing	Test data generation for embedded systems based on GA using control flow graph construction
26	Mao and Yu (2013)	5	Structural testing	Used variation of GA called Quantum inspired GA for test data generation to improve program coverage
27	Liu et al. (2013)	5	Structural testing	Test data generation using modified GA to avoid premature convergence
28	Suresh and Rath (2013a)	4.5	Structural testing	Test data generation for basis path testing using GA
29	Arcuri and Fraser (2013)	5	Structural testing	Studied the effect of parameter settings for object oriented programs using Evosuite tool and proved that parameter tuning may or may not good result. If search budget and time is a constraint, then default value of parameters may be used for such problems rather than going for parameter tuning
30	Fraser et al. (2013, 2015)	5	Structural testing	Extended the GA based Evosuite tool to a memetic algorithm based approach to improve the performance of GA during test data generation
31	Galeotti et al. (2014)	5	Structural testing	Integrating DSE(dynamic symbolic execution) approach with GA based Evosuite tool for test data generation of programs

approach. They have proposed this approach in order to eliminate the disadvantages caused by both these methods. In their work, a constraint solver is used to ensure that the offspring produced by mutation does not get stuck in local optima and they follow a different control flow. Finally, they prove that GA-DSE (Genetic algorithm based dynamic symbolic execution) has higher branch coverage than random search, GA or DSE. Khor and Grogono (2004) have used the concept of GA and formal concept analysis to generate test data for branches. They develop a tool called Genet. The authors have not used any branch functions or any program graphs for generating test data. They mentioned that the structural complexity is an unreliable measure to describe the programs and they suggested that the search space feasibility is a better indicator. They used concept analysis instead of fitness function in their approach. The concepts were ranked and the concepts with highest rank are the winner or cover more branches.

Wegener et al. (2001) used simple GA as well as multi-population GA to generate test data for structural testing of real-world embedded software systems. They compared the results and finally multipopulation GA proved to be better than simple GA and both of these proved to be better than random method (2001). Fraser et al. (2013, 2015) extended the GA based Evosuite tool to a memetic algorithm based approach to improve the performance of GA during test data generation. In their method, local search is applied at regular interval to improve the performance. The new method gave higher branch coverage than ordinary GA based method. Galeotti et al. (2014) integrated DSE (dynamic symbolic execution) approach with GA based Evosuite tool for test data generation of programs and the results of their showed that the integrated approach gave better code coverage than GA. Mao and Yu (2013) used a variation of GA called Quantum inspired GA (QIGA) for test data generation to improve program coverage. In their work, the search space is enlarged to avoid local optima. The results showed that unlike GA based approaches, QIGA has less chance to get stuck in local optima. Liu et al. (2013) used a modified GA based approach in their work. The modified method avoided premature convergence and converged faster than ordinary GA based method. The modified GA also gave higher test data efficiency compared to GA based method. Lin et al. (2012) used an adaptive GA for test case generation. Adaptive GA gave better performance compared to ordinary GA. Table 6 gives a summary of the list of works described in this paragraph. From Table 6 we can see that the variants of GA are proved to be better than simple GA.

Population representation, size and generations during software testing:- In GA based software testing, population representation, size of population and generation is an important factor (Harman 2007; Sthamer 1996). Population can be represented as a group of 0 and 1's, as a group of integers, as decimal numbers or as characters. In some problems a tree representation is also possible. Based on the problem, appropriate method of representation is applied. Improper representation of the individual in genetic algorithms may cause unexpected variations in the final result. Xue-ying et al. (2005) used a subset of test cases for population representation during test suite reduction and Xiao and Afzal (2010) used a binary representation of bug id for representation of candidate solution during bug fixation. In some other works which use genetic algorithm for structural testing, an array of bits, base 10 representation, binary representation, string of characters, integers and binary string were used to represent the population (McMinn 2004). The population size can also be a confounding factor because if the population size is too small the genetic algorithm will not search all the possible solution areas to procure an optimal solution (McMinn 2004). In this case, the individuals may reproduce abundantly and the resulting diversity in population may cause the individuals to converge to a point which appears to be better than the neighboring points.

Table 6 Works on variations of GA

Work	GA variation	Result
Xue-ying et al. (2005)	New type of GA called GeA for reducing cost associated with test suit reduction	GeA is better than ordinary GA
Latiu et al. (2012)	New approach based on simulated annealing for path testing	Simulated annealing based approach is better than GA based approach
Malburg and Fraser (2011)	Hybrid method which combines GA based and constrained test data generation approach	Hybrid method proved to have better performance
Khor and Grogono (2004)	Used concept analysis along with GA for structural test data generation	Concept analysis with GA is better than simple GA
Wegener et al. (2001)	Applied GA and multi-population GA for structural testing of real-world embedded software systems	Proved multi-population to be better than simple GA
Fraser et al. (2013, 2015)	Extended the GA based Evosuite tool to a memetic algorithm based approach to improve the performance of GA during test data generation	Proved that memetic algorithm based GA approach gives better branch coverage than GA
Galeotti et al. (2014)	Integrating DSE(dynamic symbolic execution) approach with GA based Evosuite tool for test data generation of programs	Gives better code coverage than GA
Mao and Yu (2013)	Used variation of GA called Quantum inspired GA (QIGA) for test data generation to improve program coverage	As the search space is enlarged, the new QIGA avoid local optimal solution compared to GA
Liu et al. (2013)	Used a modified GA	Modified approach gives higher test data efficiency and avoids premature convergence compared to GA
Lin et al. (2012)	Used adaptive GA for test case generation	Adaptive GA gives better performance compared to simple GA

In such a situation, even though there is a chance that a better solution exists, it is missed as the population size is already declared to be very small. This is known as the premature convergence problem ([Harman and McMinin 2010](#)).

We can see that the size of population and the number of generations were also set to different rates in the selected works. Generations decide the number of runs or the number of times the whole genetic cycle may be repeated to obtain the required solution. As it is not possible to repeat the whole process indefinitely, a stopping criterion should be defined. In other words, stopping criteria decides the cause of algorithm termination. Genetic algorithm terminates when the required result is obtained or for a specific number of generations or when there is no improvement in the fitness value which is the best in a specified time interval. Usually in GA, as the number of generation increases the end result also improves which in turn causes an increase in execution time of the whole process. Therefore population representation, size and generation are critical factors in GA based testing.

In Table 7, we have listed the type of population, size and generations used in some of the GA based software testing works. We have listed the type of population, size and generations used in them. Different population settings are used in different works. Hence, we can infer that, declaring the correct population size still remains a problem in genetic algorithm.

Operator selection and parameter settings during software testing:- In genetic algorithm based program testing, operator selection and parameter setting needs special attention. For example in the case of crossover and mutation, their rates should be not be set at either high or low levels. In selection, individuals are selected from the parent population for crossover and mutation to produce next generation individuals (Holland 1975; Pei et al. 1994; Watkins 1995). There are different types of selections like roulette wheel, tournament selection, random selection, best selection etc. In roulette wheel selection individuals are selected according to their fitness. Each individual will be assigned a fitness value and the normalized fitness value is calculated. After calculating the normalized fitness value, accumulated fitness value is calculated by adding the fitness value of the concerned individual and the sum of the fitness value of all other individuals. A random number is selected between 0 and 1 and the selected individual will have an accumulated fitness value greater than all other previous individuals but less than the remaining individuals. Tournament selection is a refinement of roulette wheel selection. Here, roulette wheel selection is repeatedly applied to produce a group of population and the best individual is selected from this group. In random selection method, the chromosome is selected randomly from the given population whereas in best selection method the individual with the highest fitness value is selected. There are many other types of selection methods, but we have mentioned only a few.

The process of crossover and mutation has serious impact on the process of test data generation using genetic algorithm. Some of the most commonly used types of crossover are one point crossover, two point crossover and uniform crossover. By using uniform crossover the diversity in the individuals produced is more compared to single and two point crossover and a better result is obtained. Mutation is the process of altering the value of genes present in the chromosome for creating genetic diversity (Goldberg 1989). Diversity in the population will create better individuals compared to a population without genetic diversity. According to the problem to be solved, mutation rates can be set to specific values. If the rate of mutation is set to high value, the search will become similar to a random search and if the mutation rate is very low then there will be no diversity in the population. From the list of works given in Table 8, we can see that different types of crossover, mutation and selection are used in the referred works. The operators used in these works and parameter settings used in these works are analyzed and listed. Different parameter settings are used in different works. Hence, from the above mentioned facts we can conclude that selecting the apt operators and setting the parameters is still an unresolved issue in GA based testing

Fitness function design during software testing:- Applying genetic algorithm in program testing requires optimizing the specified fitness function (Ali et al. 2009). A fitness function should be designed in such a way that it gives optimal solution for a given problem (Bueno and Jino 2002). Defining the fitness function imprecisely may lead to a wrong solution or may cause the problem to be stuck in the local optima (Goldberg 1989; Michael et al. 2001; Fraser et al. 2015, 2013; McMinn 2013; Suresh and Rath 2013a). The misleading nature of fitness function creates several problems. For example, the individuals with lower fitness values may be finalized as the optimal solution even when better individuals exist. This mainly occurs when the population size is smaller, because with a small sized population, the result may get converged at a faster rate than normal. Thus, in a limited population, if

Table 7 Population representation, size and generations

Identifier	Work	Type of testing	Purpose	Population representation	Population size and generations
1	Xue-ying et al. (2005).	Structural testing	Reduce cost associated with test suite reduction using GA	Subset of test cases	Not specified
2	Xiao and Afzal (2010)	Structural testing	Bug fixation	Bug id represented as binary	Size of population as 100 and 500 generations
3	Latiu et al. (2012)	Structural testing	Software path testing using GA	Array of bits	Size set as 40 and generations as 100
4	Ahmed and Hermadi (2008)	Structural testing	Test data generation for multipath testing using GA	Base 10 representation of alleles	size set 30 and generations as 100
5	Pocaitilu and Ivan (2013)	Structural testing	Automated approach for branch testing using GA	Binary representation,	Size set as 6,10, 16, 20, 26....110 and Generations as 107
6	Roper et al. (1995)	Structural testing	Used GA for testing C programs for attaining the branch coverage criteria	Character string,	Size set as 30, 80 and generations as 6, 25
7	Jones et al. (1996)	Structural testing	Branch coverage using GA	Binary representation	Size set as 45, generations not specified
8	Pargas et al. (1999)	Structural testing	Path coverage and Branch coverage using TGen tool	String of characters	Size set as 100 and the number of generations considered until the specific branch coverage is obtained
9	Michael et al. (2001)	Structural testing	Developed a tool called GADGET which uses genetic algorithms for generating test data for branch coverage of C programs	Binary string	Population size set as 24 and 100 and the number of
10	Sofokleous and Andreou (2008)	Structural testing	Test data generation framework using genetic algorithms to provide edge/partition coverage	Integers	Size set as 100 and the generations set as 600
11	Chen et al. (2009)	Structural testing	Test data generation for branch coverage	Not specified	Size set as 100
12	Berndt and Watkins (2004)	Structural testing	Studied the effect of response time in GA based testing	Not specified	Not specified

Table 7 continued

Identifier	Work	Type of testing	Purpose	Population representation	Population size and generations
13	Khor and Grogono (2004)	Structural testing	Used the concept of GA and formal concept analysis to generate test data for branches	Real value representation	Size set as 250, Generations set as 50
14	Cao et al. (2009)	Structural testing	Test data generation of a specific path using GA	Binary representation	Size = 80, generations = 800
15	Malburg and Fraser (2011)	Structural testing	Introduced a hybrid method which combines GA based and constrained based test data generation approach	Bit vector	Not specified
16	Zhang et al. (2007)	Structural testing	Test data generation of many paths using GA	Binary	Size set as 90 for one experiment and size set as 130 for another experiment
17	Fraser and Arcuri (2012)	Structural testing	Studied the effect of seeding in test data generation of object oriented programs	Method calls	Size set as 80 and the number of generations set as 10 minute time out (or 1,000,000 statements)
18	Arcuri and Fraser (2011)	Structural testing	Studied the effect of parameter settings for object oriented programs using Exosuite tool	Method calls	Size set as 4, 10, 50, 100, and 200. Generations set as 10 minute time out (or 1,000,000 statements)
19	Bueno and Jino (2002)	Structural testing	Path coverage testing using GA	Binary string	Size set as 80
20	Wegener et al. (2001)	Structural testing	Test data generator for structural testing of real-world embedded software systems using GA	Integer representation	Generation set as 200
21	Miller et al. (2006)	Structural testing	Test data generation for Branch coverage using GA	Integer representation	Population size set as 100 and generations set as 300
22	McMinn (2013)	Structural testing	Studied Impact of cross over	Input vector	Size set as 300
23	Fraser et al. (2013)	Structural testing	Test suite generation for object oriented programs which cover multiple goals simultaneously	Method calls	Size set as 80 and the number of generations set as 10 minute time out (or 1,000,000 statements)

Table 7 continued

Identifier	Work	Type of testing	Purpose	Population representation	Population size and generations
24	Gonga et al. (2011)	Structural testing	Test data generation for many paths using GA	Integer	Generations set as 70 for one experiment and as 130 for another experiment
25	Pocatu and Ivan (2013)	Structural testing	Test data generation for embedded systems based on GA using control flow graph construction	Sbyte (integer)	Size set as 100 and generations as 100
26	McMinn (2013)	Structural testing	Used variation of GA called Quantum inspired GA for test data generation to improve program coverage	Q-bit (Quantum chromosome)	Size set as 30 and generations as 100
27	Liu et al. (2013)	Structural testing	Test data generation using modified GA to avoid premature convergence	Real numbers	Size set as 15 and generations as 30
28	Suresh and Rath (2013a)	Structural testing	Test data generation for basis path testing using GA	Binary	Size set as 100 and generations as 500
29	Arcuri and Fraser (2013)	Structural testing	Studied the effect of parameter settings for object oriented programs using Evosuite tool and proved that parameter tuning may or may not give good result. If search budget and time is a constraint, then default value of parameters may be used for such problems rather than going for parameter tuning	Method calls	Size set as 4, 10, 50, 100, and 200. Generations set as 10 minute time out (or 1,000,000 statements)
30	Fraser et al. (2013, 2015)	Structural testing	Extended the GA based Evosuite tool to a memetic algorithm based approach to improve the performance of GA during test data generation	Sequence of statements of length 1	Size set as 5, 25, 50, 75, 100 and Generations set as 10 minute time out

Table 8 Operator and parameter setting

Identifier	Work	Selection	Crossover operator and rate	Type of mutation and rate
1	Xue-ying et al. (2005)	Roulette wheel selection	One point	1/L where L is the number of bits in the gene
2	Xiao and Afzal (2010)	Ratio method	0.8	0.01
3	Latiu et al. (2012)	Best fitness	One point and rate as 0.75	0.1
4	Ahmed and Hermadi (2008)	Roulette wheel	Single point and rate set as 0.5 or 0.9	0.1 or 0.3
5	Pocatitu and Ivan (2013)	Binary tournament	Two point and rate set 1.0	0.01
6	Roper et al. (1995)	Random selection	Single, double and uniform crossover point	Simple mutation.
7	Jones et al. (1996)	Random selection	One point	Simple mutation, Mutation rate decide by user
8	Pargas et al. (1999)	Random selection	One point	Simple mutation and and rate as 0.10
9	Michael et al. (2001)	Random selection	One point	Simple mutation and rate set as 0.001
10	Sofokleous and Andreou (2008)	Tournament	Inter and Intra crossover	Inter and intra mutation
11	Chen et al. (2009)	Random	0.4	0.1
12	Berndt and Watkins (2004)	Not specified	Not specified	Not specified
13	Khor and Grogono (2004)	Selection based on the ranking of individuals according to the concepts	Uniform crossover, rate = 0.8	Mutation rate = 0.4
14	Cao et al. (2009).	Roulette wheel	0.80	0.15
15	Malburg and Fraser (2011)	Random	Several setting	Several settings
16	Zhang et al. (2007)	Roulette wheel	One point, rate is 0.9	One point mutation and rate = 0.3
17	Fraser and Arcuri (2012)	Rank selection based on fitness	Special type of one point crossover where the first part of the sequence of statements of the 1st parent is merged with the second part of the second parent and vice versa	Mutation probability for test suite = 1/T, where T is the test suite Mutation probability for test case = 1/3, where the operations applied are remove, change and insert

Table 8 continued

Identifier	Work	Selection	Crossover operator and rate	Type of mutation and rate
18	Arcuri and Fraser (2011)	Roulette wheel, Tournament and Rank selection	Special type of one point crossover where the first part of the sequence of statements of the 1st parent is merged with the second part of the second parent and vice versa	Mutation probability for test suite = $1/T$, where T is the test suite Mutation probability for test case = $1/3$, where the operations applied are remove, change and insert
19	Bueno and Jino (2002)	Individuals selected based on Previous knowledge	Single point	Simple and rate set as 0.03
20	Wegener et al. (2001)	Individual selected based on fitness	One point	Discrete recombination (Different mutation range for each subpopulation)
21	Miller et al. (2006)	Tournament	One point	Uniform random mutation, non-uniform random mutation and Muhlenbein's mutation, is randomly chosen
22	McMinn (2013)	Best fitness	One point, uniform and discrete recombination	Breeder genetic algorithm mutation operator applied at an inverse of chromosome length
23	Fraser et al. (2013)	Rank selection based on fitness	Special type of one point crossover where the first part of the sequence of statements of the 1st parent is merged with the second part of the second parent and vice versa	Mutation probability for test suite = $1/T$, where T is the test suite Mutation probability for test case = $1/3$, where the operations applied are remove, change and insert
24	Gonga et al. (2011)	Roulette wheel	One point, rate is 0.9	One point, rate is 0.3
25	Pocatlil and Ivan (2013)	Roulette wheel	0.1	0.8
26	McMinn (2013)	Gambling roulette selection	One point, rate is 0.90	Quantum rotation gate technique, rate is 0.05
27	Liu et al. (2013)	Not specified	0.8	0.4
28	Suresh and Rath (2013a)	Fitness based selection	Two point, rate is 0.5	Bit wise, rate is 0.05

Table 8 continued

Identifier	Work	Selection	Crossover operator and rate	Type of mutation and rate
29	Arcuri and Fraser (2013)	Roulette wheel, Tournament and Rank selection	Special type of one point crossover where the first part of the sequence of statements of the 1st parent is merged with the second part of the second parent and vice versa	Mutation probability for test suite = $1/T$, where T is the test suite Mutation probability for test case = $1/3$, where the operations applied are remove, change and insert
30	Fraser et al. (2013, 2015)	Rank selection based on fitness	Special type of one point crossover where the first part of the sequence of statements of the 1st parent is merged with the second part of the second parent and vice versa. If $P1$ and $P2$ are parents then $O1 = \alpha P1$, $(1 - \alpha)P2$ and $O2 = \alpha P2$, $(1 - \alpha)P1$, where $O1$ and $O2$ are offspring and α is a random value chosen from $[0,1]$	Mutation probability for test suite = $1/T$, where T is the test suite Mutation probability for test case = $1/3$, where the operations applied are remove, change and insert

one of the individuals surpasses the neighboring individuals, then that point or individual will be considered as the best solution even when better solutions exist. Considering these local points as the candidate solutions and assigning higher fitness values to them will result in a diversion from the original solution. This results from the inherent weakness of genetic algorithms (Michael et al. 2001). A group of researchers used an evolutionary algorithm along with a reprogrammable hardware array and the fitness function was designed to output an oscillating signal (Graham-Rowe 2002). At the final stage of the experiment, the researchers found that the circuit had become a radio receiver which was able to pick up and relay an oscillating signal from the nearby electronic device. Here, there was a deviation from the main goal itself and this was due to the fault in the design of the fitness function (Graham-Rowe 2002). Each of the works which use genetic algorithm for software testing has designed their own fitness function (Ali et al. 2011). For example, we can see that in the path testing, fitness function can be a measure of the function of branch distance and approximation level or sum of normalized intermediate fitness functions of multiple paths for multipath testing or a measure of weighted hamming distance of predicate value and the reciprocal of the branch predicate value (Arcuri and Fraser 2011; Pocatilu and Ivan 2013). Similarly, in model based testing fitness function can be a measure of test paths covered by the chromosome or measure of proximity to the constraints satisfied or measure of dependencies between components (Dongsa-ard et al. 2007). Here, we have mentioned only a few possibilities.

Table 9 given above explains the fitness function design strategies used in various works which uses GA for software testing. We have identified the works according to the type of testing and coverage criteria. After that the fitness function design strategy is listed for each work. From the above facts we can conclude that, designing fitness function deserves prime importance in GA based testing.

2.4 Discussion and review result/observation

This is the final stage in our review. In Sect. 2.3.1, we saw the primary sources in our review which was selected based on the data extraction criteria. We can notice that different population representation, operator and parameter settings and fitness function was used in GA based software testing. We have made a structured ordering of some of the most relevant observations in the referred works for answering the research questions. Finally, we have arrived at some conclusion by referring some works in GA based software testing which are listed in Sect. 2.3.1.

RQ1 In spite of the large volume of works in genetic algorithm based testing, why some works have considered variations of genetic algorithms? Even when a group of researchers claim that simple GA is best for software testing, we can see that a lot of works have used several variations of GA for software testing. In Section 2.3.1, the list of the works which use variation of GA in software testing is given and the observations are given in Table 10. Most of these works shows that variations of genetic algorithm perform better compared to simple GA.

Form the list of observations and data collected from various works, we can see that till now the researchers are not able to conclude which type of GA or what variation of GA is best for various software testing strategies.

Future research directions:- From the list of works which uses GA variation for software testing, we can recommend that the researchers should try to focus research in the direction which is represented in Fig. 6.

Table 9 Fitness function calculation

Identifier	Work	Coverage	Fitness function
1	Xue-ying et al. (2005)	Test suite coverage	Measure of coverage and cost
2	Xiao and Afzal (2010)	Test data for covering bugs	Value(B) = $\alpha \cdot \text{priority} + \beta \cdot \text{severity}$ * Has finished(B), where B is the bug and α and β are the weights for priority and severity, Has finished (B) may have value 1 or 0 if the bug is fixed before deadline and if the bug cannot be fixed
3	Latiu et al. (2012)	Path coverage	Function of branch distance and approximation level (Branch distance designed based on Korel's branch distance function)
4	Ahmed and Hermadi (2008)	Multipath coverage	FF = Sum of normalized intermediate fitness functions of multiple paths
5	Pocatlil and Ivan (2013)	Branch coverage	FF = Approximation level + Normalized branch distance
6	Roper et al. (1995)	Branch coverage	Two fitness function-First one is the weighted hamming distance of predicate value, Second one is the reciprocal of the branch predicate value
7	Jones et al. (1996)	Branch coverage	Measure of coverage
8	Pargas et al. (1999)	Path coverage and Branch coverage	Measure of number of common branch predicated in the control flow graph of the program
9	Michael et al. (2001)	Branch coverage	Expressed as Predicate function based on Korel's fitness function
10	Sofokleous and Andreou (2008)	Edge/partition coverage	For BO-GA, Fitness function $F = w1(\#edges\ exec) + w2(\#predtrue + \#predfalse)/w1 + w2$, where $w1$ and $w2$ are the weights in the range [0,1] and $\#edges\ exec$ is the number of executed edges and $\#predtrue$ and $\#predfalse$ is the number of simple predicates evaluated at least once to true or once to false
			For CU-GA, fitness function $F_{close-up} = F_{vert} + 1/F_{dist}(c)$, where F_{vert} is the sum of exercised vertices and $F_{dist}(c)$ focus on the condition that prohibits test case to visit next vertex
11	Chen et al. (2009)	Branch coverage	Set by analyzing the pre-dominator tree used to construct the EPDG
12	Berndt and Watkins (2004)	Not mentioned	Not mentioned
13	Khor and Grogono (2004)	Branch coverage	Uses concept analysis
14	Cao et al. (2009)	Path coverage	Concept pair = (o, a), where o is the object set and a is the attribute set Measure of similarity between target path and execution path with sub path overlapped

Table 9 continued

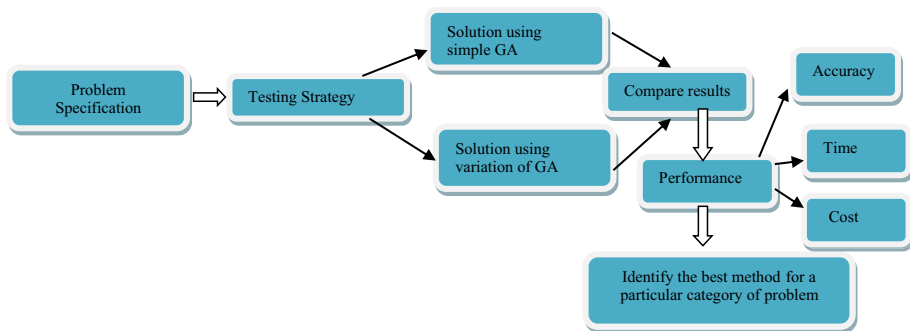
Identifier	Work	Coverage	Fitness function
15	Malburg and Fraser (2011)	Branch coverage	Integrates approach level (number of unsatisfied control dependencies) and branch distance
16	Zhang et al. (2007)	Path coverage	Measure of approach level and branch distance level
17	Fraser and Arcuri (2012)	Branch coverage	Fitness (T) = $ M - M_T + \sum_{b_k \in B} d(b_k, T)$, where T is the test suite, M is the methods in the program and MT is the number of executed methods and d(bk, T) is the branch distance for branch b on test suite T
18	Arcuri and Fraser (2011)	Branch coverage	Fitness (T) = $ M - M_T + \sum_{b_k \in B} d(b_k, T)$, where T is the test suite, M is the methods in the program and MT is the number of executed methods and d(bk, T) is the branch distance for branch b on test suite T
19	Bueno and Jino (2002)	Path coverage	FT = NC-EP/MEP
20	Wegener et al. (2001)	Statement and branch coverage	Measure of approximation level and normalized predicate level distance
21	Miller et al. (2006)	Branch coverage	Rules to build fitness function for basic relational operations Example Fitness function (where p is a preset penalty value and a and b are expressions):- if (a), F = 0 for true and F = p for false if (a = b), F = 0 for (a = b) and F = abs(a - b) + p for a $\neq$ b if (a < > b), F = 0 for (a = b) and F = p for a $\neq$ b if (a < b), F = 0 for (a < b) and F = abs(a - b) + p for a $\geq$ b if (a <= b), F = 0 for (a $\leq$ b) and F = abs(a - b) + p for a > b
22	McMinn (2013)	Branch coverage	Fitness (T) = $ M - M_T + \sum_{b_k \in B} d(b_k, T)$, where T is the test suite, M is the method in the program and MT is the number of executed methods and d(bk, T) is the branch distance for branch b on test suite T Represented as an n-dimensional vector, where each dimension is related with a target path and the fitness function of a target path is given as, FF = Approximation level + Normalized branch distance
23	Fraser and Arcuri (2012)	Branch coverage	
24	Gonga et al. (2011)	Multipath coverage	

Table 9 continued

Identifier	Work	Coverage	Fitness function
25	Pocatuil and Ivan (2013)	Path coverage	Fitness function is measured as inverse similarity coverage (ISC), $F = \text{len}(a)/\text{sym}(a)$, where $\text{len}(a)$ gives the length of the path given by chromosome a and $\text{sym}(a)$ is the similarity function
26	Mao and Yu (2013)	Branch coverage	Measure of branch distance function for the i th in a program ($bchi$) and the corresponding branch weight (w_i) $1 \div \left[\theta + \sum_{i=1}^s w_i \cdot f(bchi_i) \right]$
27	Liu et al. (2013)	Branch and path coverage	Ratio of branch (f) or path of practical coverage of test case to the total branches (z) or paths, $F = f/z$
28	Suresh and Rath (2013a)	Path coverage	Fitness function is a measure of branch distance function based on Korel's branch distance function
29	Arcuri and Fraser (2013)	Branch coverage	Sum of the normalized branch distances of all branches in the program under test
30	Fraser et al. (2013, 2015)	Branch coverage	Sum of normalized minimal branch distance

Table 10 Observations on works using GA variation for software testing

Number of works using GA variation for test data generation	Number of works in which variation of GA is proved to be better than GA	Number of works in which GA is proved to be better	Comments
9	9	NIL	Since variation of GA is proved to be better than GA in all works, it's high time to assess which variant is best in software testing

**Fig. 6** Research direction suggested from the observations in GA variation

What we can infer from all this is that, depending on the system under test/problem, the type of GA or the variation of GA used also differs. This is one of the areas which future researchers should concentrate upon. They should clearly elucidate which type of GA or what variation of GA is universally applicable to a particular category of problems during software testing. For example, a researcher should be able to identify or get a clear idea of what type of GA or GA along with some method or any variation of GA gives the best result during model based testing, structural testing, stress testing, mutation testing etc. Little works have shown interest in this issue till date. This may be due to wide possibilities or variations of GA which may be applied during software testing. We recommend that, trying to perform such an in-depth study will clear the air regarding the use of GA in software testing to a great extent. We suggest that the researchers should try to focus research in the direction given below.

Advantages of suggested research focus:- Programmers will be confident of the type of GA/GA combined with any other method/GA variation may be apt for a particular category of problem

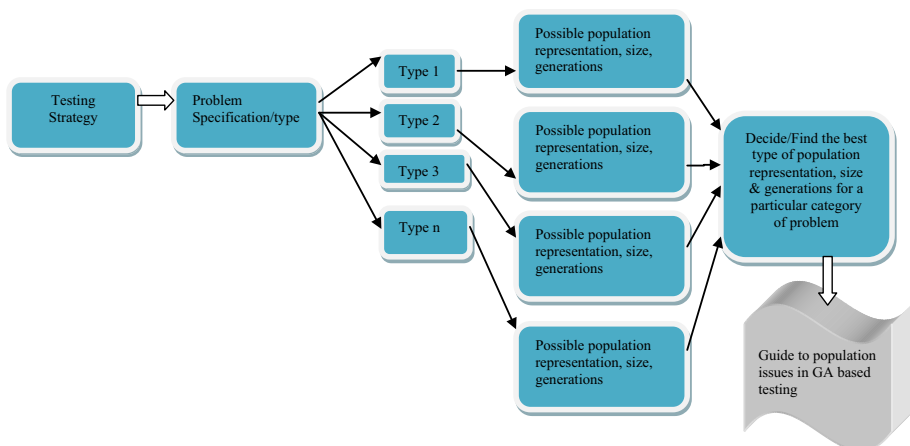
- Unnecessary effort and time spend to find variants of GA in software testing may be reduced.

RQ2 What is the effect of population representation and size in software testing? We have seen different types of population representation and different population size and the number of generations used in software testing from the Table 7 given in Sect. 2.3.1. Some observations from Table 7 are given below in Table 11.

Table 11 Population representation for structural testing

Number of works using different Population representation for structural testing

Binary	Base 10	Character	Array of bits	Integer	Real	Sbyte	Qbit
8	1..	1	2	1	2	1	1

**Fig. 7** Research direction suggested from the observations in population representation in GA based software testing

From Table 11, we can see that for structural testing several types of population representation are used.

Future research directions:- From the list of observations on population representation, size and number of generations in GA based software testing, we can recommend that the researchers should try to focus research in the direction which is represented in Fig. 7.

In the Fig. 7, we can see that, for a given problem specification, we suggest to find the most suitable type of population representation, size and generation. In most of these works it is mentioned that they have chosen a particular type of representation for population by referring previous works, whereas in some works it is mentioned that they have chosen a particular representation randomly. Only a very few works have given a clear explanation of choosing a particular type of population representation for the system under test. For example, Fraser et al. have claimed that, seeding strongly influences the efficiency of search based test data generation and they have used seeding to improve the process of population initialization (Fraser and Arcuri 2012). Similar is the case with population size and the number of generation. Here, if the researchers are able to get a picture of the different types of population representation and the approximate size of population according to the type of software testing, the issues related to the size and representation of the population may be solved. For example, if a study is conducted to finalize the most suitable type of population representation for structural testing of a particular category of problem or what type of population representation is most suitable during black box testing or what type of population representation is best during model based testing, software testers may not have any confusion in population representation. Instead of adopting a particular type of population representation, size and generations by referring previous literatures, researchers should try to find a general or most

suitable type of population representation, optimal value of size and number of generations for a particular category of problem during various testing strategies. There lies immense research scope in this direction. Researchers have not explored this direction yet. Instead, they have randomly selected a particular type of population representation, size and generation and proceeded directly to the test data generation process. Testing should be done using different population settings and the best population setting should be reported. This in turn may resolve the uncertainties prevailing in population settings during GA based testing.

Advantages of suggested research focus:-

- Problems related to population representation, size and generation in GA based software testing may be resolved to a great extent if general guidelines are formed for population issues.
- Researchers will be confident on the method selected if the base values are set according to some general guidelines

RQ3 Is there any common method to design fitness function during software testing? Fitness function is one of the core aspects of GA based testing. The result of GA based testing depends on fitness function. Design of fitness function differs according to the type of testing, purpose or coverage and the method used for designing the fitness function. For example, before using GA for structural testing, tester should have an idea about how to design fitness and the factors to be considered for designing fitness function. This is because, many factors such as program dependency and path selection affects the process of fitness function design. After designing the fitness function, parameters are tuned to get the expected result. Response time of the system is also in turn dependent on parameter tuning. Therefore, setting up general guidelines for fitness function design in GA based testing minimizes the issues related to parameter setting in GA. From Table 9 given in Sect. 2.3.1, we can see that several approaches are taken for designing the fitness function according to the coverage criteria. One of the most important factors to be considered during fitness function design is the program dependency consideration. In most of the genetic algorithm based software testing, program dependency is not correctly followed (Harman and McMinn 2010; McMinn 2005). An evidence of this fact may be drawn from the Table 12 given below which lists the future scope of the referred works. In Table 12, the factors to be resolved in future are extracted from the referred papers. In Table 12, we can see that most of the works have mentioned to solve dependency related issues in their future work. Solving dependency related issues in turn depends on the path identification. We can notice that most of the works have also reported path identification problem as their future research perspective. If there is no automated method to identify the potential paths during fitness function design, all the statements in the program should be analyzed to identify the relevant statements. In GA based software testing, only a very few research works have addressed the problem of potential path identification (Harman et al. 2007).

Even though fitness function design varies according system under test, general guidelines for designing the fitness function based on the category of system under test or the factors to be considered during fitness function design may be established by the researchers. Though a very few works have mentioned about such possibilities, this problem is yet to be discussed deeply. If in future, such a study is accomplished, it will go long way in making the use of GA based software testing applicable to all type of system irrespective of whether the system is large or small. Unfortunately, little attempt is made for such a study, whereas, most of the effort is spent on designing newer fitness function for testing. Figure 8 given below, gives the

Table 12 Future issues to be solved in referred works

Identifier	Work	Type of testing	Factors to be resolved in future
1	Xue-ying et al. (2005)	Structural testing	Effect of different operators and parameter setting may be studied Optimal/best parameter setting may be found by conducting more experiments Dependency between different test cases may be discussed and the effect of applying metaheuristic on the same
2	Xiao and Afzal (2010)	Structural testing	Operator and Parameter setting may be studied in detail The side effects of fixing the bugs may be discussed. In other words, the solving the dependency issue between the bugs
3	Latiu et al. (2012)	Structural testing	To find the reason why GA based method is less competing than PSO and SA Study parameter settings
4	Ahmed and Hermadi (2008)	Structural testing	How to identify target paths Study the dependency issue between multiple paths during testing Study operator and parameter settings
5	Pocutilu and Ivan (2013)	Structural testing	How to identify target paths Study the dependency issue between multiple paths during testing Study operator and parameter settings
6	Roper et al. (1995)	Structural testing	Solving dependency issues Operator and parameter setting variations
7	Jones et al. (1996)	Structural testing	Scalability of the approach Solving dependency issues Operator and parameter setting variations
8	Pargas et al. (1999)	Structural testing	The authors claim that their approach is scalable. Experiments are needed to prove their claim Covering full dependency in the program using a program dependence graph instead of control flow graph Operator and parameter setting variations
9	Michael et al. (2001)	Structural testing	Method to identify the predicates should be found out

Table 12 continued

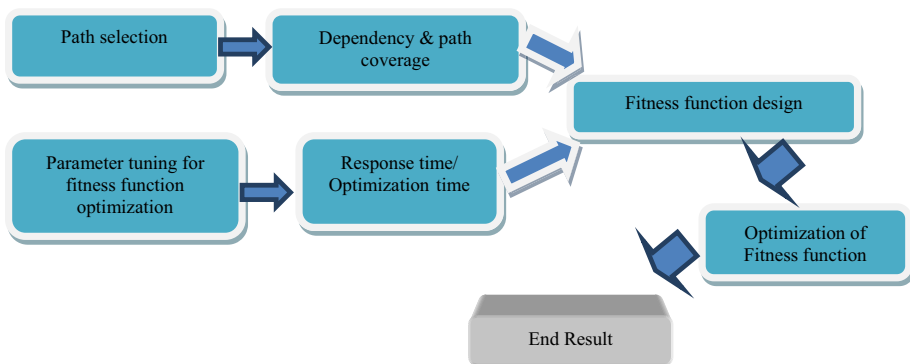
Identifier	Work	Type of testing	Factors to be resolved in future
			Methods to prioritize branches during testing
			Solving the dependency issues in the program
			Operator and parameter setting variations
			As the tool is developed for C program,
10	Sofokleous and Andreou (2008)	Structural testing	The authors claim that their approach is scalable. Experiments are needed to prove their claim
			Covering full dependency in the program using a program dependence graph instead of control flow graph
			Operator and parameter setting variations
11	Chen et al. (2009)	Structural testing	Solving dependency issues and finding rules for setting parameters and measures needed to improve fitness function
12	Berndt and Watkins (2004)	Structural testing	May study operator setting variations
13	Khor and Grogono (2004)	Structural testing	Even though, the authors claim that their approach does not use any flow graph, they have also mentioned that it is difficult to cover nested predicates using their approach. This issue is ought to be solved in future
14	Cao et al. (2009)	Structural testing	Automatic selection of path from control flow graph
			Study the influence of different type of operators and parameter setting
15	Malburg and Fraser (2011)	Structural testing	Methods to improve the coverage using different options of fitness function
16	Zhang et al. (2007)	Structural testing	Full automation
			Finding suitable value of Subpopulation size
17	Fraser and Arcuri (2012)	Structural testing	Study different settings for parameters and operators during test suite generation
			Study the effect of seeding for other search based techniques
18	Arcuri and Fraser (2011)	Structural testing	May extend the work to different languages
19	Bueno and Jino (2002)	Structural testing	Study operator and parameter settings
			Better methods to solve random variations

Table 12 continued

Identifier	Work	Type of testing	Factors to be resolved in future
20	Wegener et al. (2001)	Structural testing	Assumes that the target is already given Multipath coverage not discussed Study the dependency issue in the program when using multiple subpopulation during testing Study operator and parameter settings
21	Miller et al. (2006)	Structural testing	Handle complex data structures like arrays, improve scalability and improve path coverage Study operator and parameter settings
22	McMinn (2013)	Structural testing	May conduct study on various type of systems to study the impact of crossover, so that the method may be generalized
23	Fraser and Arcuri (2012)	Structural testing	Methods to handle collateral coverage of individuals (As the overhead increases when collateral coverage is considered) Study the effects of different coverage Test the approach using other type of algorithms like simulated annealing in addition to genetic algorithms
24	Gonga et al. (2011)	Structural testing	Full automation of the approach As the initial population is split-up into subpopulation, assigning correct size to the subpopulation remains an issue Study operator and parameter settings
25	Pocatilu and Ivan (2013)	Structural testing	Fitness function improvement Compare the performance with other genetic algorithm based test data generation methods
26	Mao and Yu (2013)	Structural testing	Fitness function improvement Improve coverage of the method
27	Liu et al. (2013)	Structural testing	Improve time efficiency (Parameter tuning and parameter selection)
28	Suresh and Rath (2013a)	Structural testing	Improve path coverage Handle complex program

Table 12 continued

Identifier	Work	Type of testing	Factors to be resolved in future
29	Arcuri and Fraser (2013)	Structural testing	Study the effect of parameter tuning to different kind of problems [As parameter tuning may/may not result in worse result compared to the result obtained without parameter tuning]
30	Fraser et al. (2013, 2015)	Structural testing	Find optimal parameter configurations as the result of the suggested work depends on the class on which test data generation is applied
31	Galeotti et al. (2014)	Structural testing	Improve program coverage Improve performance of the method

**Fig. 8** Factors affecting and affected by fitness function design**Table 13** Main issues in fitness function design and their possible solutions

Issues in fitness function design	Suggested solutions
Trace the potential paths/relevant paths for designing fitness function	Use methods to attain maximum coverage, identify potential paths and trace dependency while designing fitness function
Trace all possible dependencies in the program so that test data generation using GA may be applied in all practical situations	
Try to attain maximum coverage during test data generation	

prime reason for why the design fitness function is considered as the most important factor in GA based software testing. From Fig. 8, we can arrive at a conclusion which summarizes the steps to be taken while designing the fitness function during GA based testing. Table 13

Fig. 9 Main steps to be followed during fitness function design

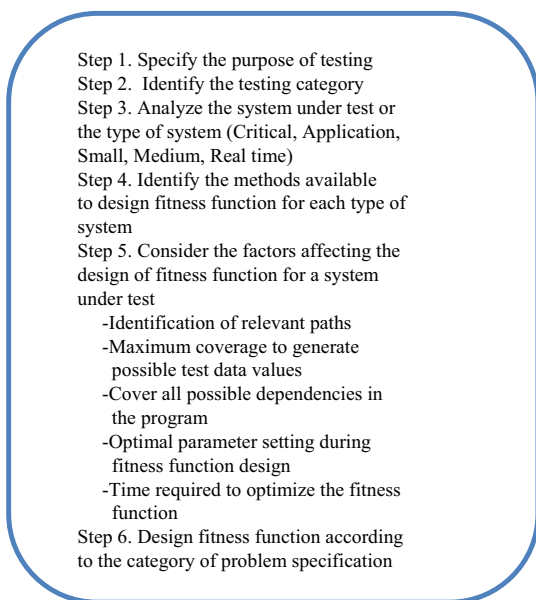


Table 14 Structural testing: number of works using different types of selection in Table 8

Tournament	4	Roulette wheel	8	Ratio	1	Binary tournament	1	Random selection	6	Rank based selection	8	Gambling roulette wheel	1
------------	---	----------------	---	-------	---	-------------------	---	------------------	---	----------------------	---	-------------------------	---

gives a gist of our conclusion on fitness function design issues. Some of the future research perspectives in fitness function design are also mentioned below.

Future research direction:- From Table 13, we saw the issues in fitness function design and their suggested solution. Therefore, we suggest the future researchers to give more emphasis on finding general methods for fitness function design according to the category of problem specification. The main steps to be followed during this process are given in Fig. 9. In Fig. 9, general strategy for designing fitness function for a given problem specification is listed. The advantage of using such a general strategy for fitness function design is given below.

Advantages of suggested research focus:- One of the main issues in using GA for practical testing may be solved by alleviating the difficulties faced during the design process of fitness function.

RQ4 What is the general strategy adopted in operator selection and parameter settings during software testing? Finding an optimal value of parameters is one of the most sought issues in GA based testing which haven't received much attention till now. In Table 8, the operators used and the parameter settings used in various works are shown. From Table 8 we can notice that, even though several works which explain different types of operators and their relevance in different contexts exist, use of these operators in specific context still remains unexploited. Table 14 shows an evidence of this factor. In most of the works we can notice that, the operators and parameters are set randomly or they have used specific operators and

parameter settings by referring some similar works in their field. Few works by researchers like [Fraser and Arcuri \(2011a, b\)](#) and [McMinn \(2013\)](#) has considered such a possibility of studying the parameter setting during GA based software testing. Even these researchers claim that more studies are required to reach a concrete conclusion. In GA based testing, even after testing a program using the best available genetic operators and parameters, a better solution or the same solution can be obtained even if we use less competing methods of crossover, selection and mutation for solving the same problem. This shows the uncertain nature of genetic algorithms [Goldberg \(1989\)](#). Most of the works given in Table 8 have mentioned about this risk. Another issue involved in parameter settings is the time taken for optimizing the fitness function. Fitness function optimization is a heuristic process and the optimization time and effort varies according to the nature of the problem ([Ahmed and Hermadi 2008](#); [Berndt and Watkins 2004](#)). Therefore, the exact time required for testing a program cannot be accurately predicted [Arcuri and Fraser \(2011\)](#). The time varies as the parameter settings are changed.

Another factor which has a major role in parameter tuning is the ‘search budget’ [Arcuri and Fraser \(2013\)](#). As budget plays a critical role in software testing, parameter tuning should be a factor of budget. From Table 12 we can see that most of the works have reported operator selection and parameter tuning as their future enhancement. It is surprising that little attempts were made on how to overcome the issues in operator and parameter settings. Instead, these issues related to parameter setting are shifted to “Threat to validity” in most of the works. One reason for this could be the difficulties in carrying out and designing such a study. Using GA based testing, even though a problem may be solved in less time, the field is still dormant to decide the best possible combination of parameter settings suitable for a given problem. This leaves immense possibilities for research in this field.

Future research directions:- Given a problem specification, although it may not be possible to find the exact value for all parameters, we suggest the researchers to look into the future issues in parameter settings which are given in Fig. 10. In Fig. 10 we can see that, after identifying the properties of the system under test, the next step is the design of fitness function. This is already visible from Fig. 8. Therefore, for finding the optimal value of parameters, the primary necessity is a fitness function, as parameters setting are used to optimize the fitness function. After finding the fitness function, the next step is to find the optimal value of parameters needed to optimize the function. For this, we have to analyze the outcome of applying basic and advanced operators during software testing. Finally, the best possible operator and parameter settings for a given category of problem may be found out. After finding the optimal parameter settings, the relation between fitness function optimization and time taken for optimization as well as the relation between fitness function optimization and search budget may be identified. A very few works have even suggested such a possibility and therefore, there lies a great research scope in this direction ([Arcuri and Fraser 2011](#); [Fraser et al. 2013](#); [Campos et al. 2014](#)).

Advantages of suggested research focus:-

- General guidelines for setting the parameters according to the problem
- Time and effort spend in parameter tuning may be minimized
- Solves the ambiguities in setting parameters during GA based testing.

Finding the convergence criteria and computing time is an important step GA based testing. Convergence defines the point at which the fitness function reaches an optimal solution

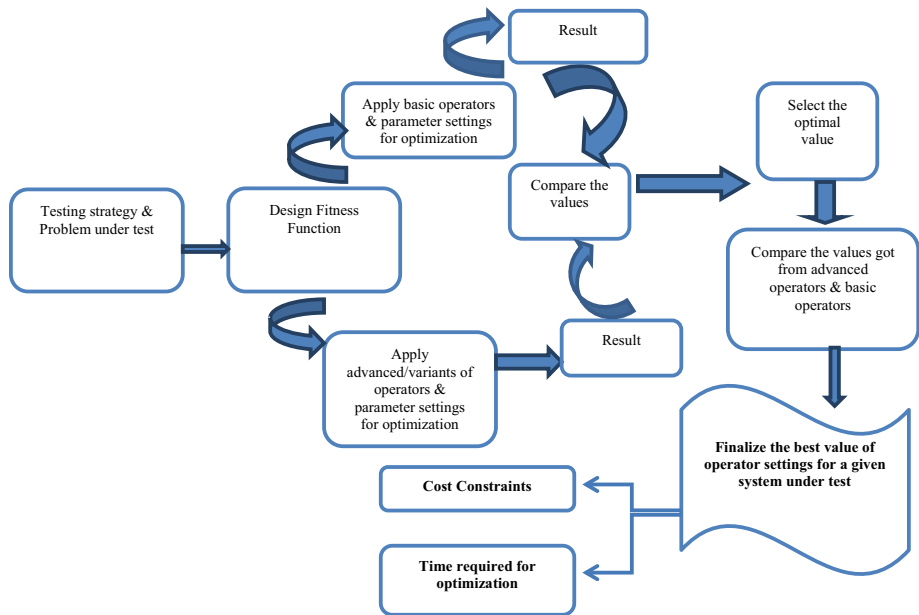


Fig. 10 Suggested research directions in parameter settings

(Holland 1975). In GA based software testing, the execution proceeds until the fitness function converges. Execution stops when the solution converges to a designated point. Sometimes, the fitness function may not converge and in such situations it is difficult to find test data. In such cases an optimal value of test data is considered. In GA based software testing, sometimes the solution converges in a fewer number of generations or after a large number of generations or may not converge itself. In such situations, inorder to avoid infinite number of executions and to finalize the optimal value of fitness function, stopping criterion must be predefined. Most common stopping criterion is the 'number of generations' required to find the test data during software testing. This is a trial and error procedure. Sometimes the target paths are infeasible and the search for test data to cover such paths never succeeds. In such situation the optimal value of test data obtained for a specified number of generations are considered. Generations decide the number of runs or the number of times the whole genetic cycle may be repeated to obtain the required solution. In Table 7, the value of generations used in various works is given. As it is not possible to repeat the whole process indefinitely, a stopping criterion should be defined. In other words, stopping criteria decides the cause of algorithm termination (Hermadi et al. 2014). Genetic algorithm terminates when the required result is obtained or for a specific number of generations or when there is no improvement in the fitness value which is the best in a specified time interval. Usually in GA, as the number of generation increases the end result also improves which in turn causes an increase in execution time of the whole process. Therefore generation is critical factors in GA based testing.

Computing time is an important factor which influences the process of test data generation in GA based software testing (Goldberg 1989). Computing time is the time taken by the fitness function to reach an optimal solution or the time taken for convergence. Usually in GA based testing as time increases, the quality of solution obtained also increases. But

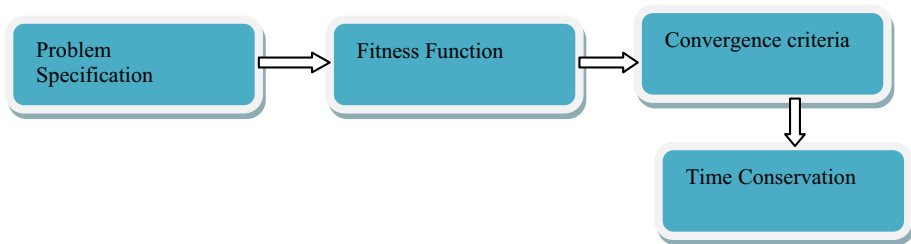


Fig. 11 Suggested research directions in convergence criteria

it cannot be assured to get a better solution in all situations. The execution may continue infinitely without converging to a solution. Therefore, in GA based testing it is better to set the convergence criteria as a certain number of generations instead of a fixed time.

Future research directions:- Finding the computation time or convergence criteria for a given type of problem may be considered as future research direction. In Fig. 11, the suggested research directions in convergence criteria are given. If the testers get an idea about the number of generations in which the fitness function convergence occurs for a given class of problem, unnecessary time wastage can be reduced. Unnecessary program execution may be avoided with the hope of getting better result in the successive generations.

Advantages of suggested research focus:-

- Avoid unnecessary computation
- Save computation time.

RQ6 What is the role of coverage in GA based testing? This is one of the important aspects of GA based testing. Various coverage criteria used in different works are given in Table 9. In GA based test data generation, the solution sought by GA is test data. The test data obtained should cause the execution of statements, branches or paths in the program. During program testing, the main aim is to attain maximum coverage so that the test data generated executes all the constraints present in the program. In some scenarios, the test data generated will not cover the infeasible paths present in the program. Unnecessary effort is wasted in exploring such infeasible paths. Therefore, in some programs, a control flow coverage criterion is considered. Control flow coverage is based on the control flow of the program. Control flow coverage may not give relation between statements in a program. Attaining a complete coverage of the program will be difficult in such situation. Therefore, data flow coverage criteria is used in some works. In data flow coverage, data dependence analysis is used to identify the relation between the statements and paths which leads to solution. Based on the coverage criterion, the fitness function is designed which leads to solution. Therefore coverage plays an important role in GA based software testing.

For a given type of problem, we should try to find out the best coverage criteria. For simple problems, statement or branch coverage will be sufficient. For large and complicated problems data flow coverage may be appropriate. Researchers should try to find out the appropriate coverage criteria according to the type of problem. After finding the best type of coverage for a particular class of problems, general guidelines to be followed for designing the fitness function is to be devised.

Future research directions:- For a given type of problem, we should try to find out the best coverage criteria. For simple problems, statement or branch coverage will be sufficient. For large and complicated problems data flow coverage may be appropriate. Researchers should try to find out the appropriate coverage criteria according to the type of problem. After finding the best type of coverage for a particular class of problems, general guidelines to be followed for designing the fitness function is to be devised.

Advantages of suggested research focus:-

- If the tester is having an idea of the type of coverage to be used for a particular class of problem, unnecessary effort spent in exploring the unfeasible paths may be prevented
- Simplifies the fitness function design process

3 Can GA based software testing evolve as one of the strongest techniques in software testing industry? If so, what are issues to be sorted out in GA based testing

This question forms the heart of this review and the answer to this questions lies within the observations made from GA variation, population, generation, fitness function design, operator and parameter settings and coverage criteria given in the previous sessions. Solving the problems mentioned in RQ1, RQ2, RQ3, RQ4, RQ5 and RQ6 may make GA based testing as one of the strongest methods in software industry If the researchers are able to find solution for the future issues mentioned in RQ1, RQ2, RQ3, RQ4, RQ5 and RQ6 the uncertainties which exist in using GA based methods in practical software testing may be eliminated.

4 Experimental evaluation: software testing using genetic algorithm

As an evidence to the observations made in this review, we have carried out an experimental evaluation of GA based software testing using GA with appropriate pre-defined parameter and without using any predefined parameters. The result from the evaluation finally support our observations made in this review.

In the experimental evaluation, we have selected some test subjects and have used GA to generate test data for these programs. One set of programs are tested using GA with pre-defined parameter settings and another set of programs are tested using GA without pre-defined parameter settings.

A research question (RQ) was formulated and this question was answered by the experimental study. The question is given below:

RQ. How effective is software testing using GA with pre-defined parameters settings compared to testing with GA without any pre-defined parameter settings?

In order to address research question RQ, the null hypothesis and alternate hypothesis were formulated as given below:-

H₀ There is no difference in the difficulty level of software testing using GA with pre-defined parameters settings compared to testing with GA without pre-defined parameter settings?

H_A Software testing using GA with appropriate pre-defined parameters settings is easier compared to testing with GA without any pre-defined parameter settings,

Table 15 Subject programs

Test subject	Description of the program	LOC
P1	A C program to find the value of y in the function y (x, n)	22
P2	A C program to calculate the commission for a sales representative	21
P3	A C program for deciding grades for the marks obtained	18
P4	A C program to calculate the bank operations	121
P5	A C program to find the grade of steel	35
P6	A C program to find the option values in online trading system	117
P7	A C program to generate Fibonacci series	20
P8	A C program to find the sum of the given series	16
P9	A C program to find biggest of three numbers	18
P10	A C program to find electricity charges	22

Some sample programs (test subjects) were considered. The specifications of the program are given below in Table 15.

In order to handle the null hypothesis H_0 , evaluation was carried out in two stages. In the first stages, test subjects P1 to P5 was tested using GA without any pre-defined parameter settings and test subjects P6 to P10 was tested using GA with pre-defined parameter settings. The programmer had to find the parameter settings by trial and error or by referring other similar condition for test subjects P1 to P5. Here, we can see that there are two test conditions and two groups of test subjects. Therefore, a Mann–Whitney U test is used to evaluate the results experiments.

During software testing, a questionnaire was prepared and was distributed to the software testers. There were 2 questions Q1 and Q2 in the questionnaire which are given below.

Q1. What is the difficulty faced due to undefined parameter settings in GA based testing?

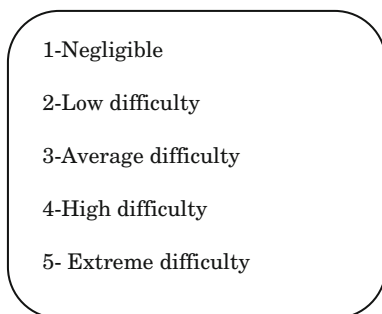
Q2. Given no guidelines, what is the problem faced in fitness function design during GA based testing?

All the test subjects (P1–P10) addressed this questionnaire. The outcome was recorded for subject programs P1–P5 for GA based testing without any pre-defined parameter settings and using GA with appropriate pre-defined parameter settings for subject programs P6–P10.

A qualitative rating for the questions was assigned based on the evidence collected from GA based testing with appropriate pre-defined parameter settings and without pre-defined parameter settings. Each question had a possible response. By data coding, the outcome of the questions was rated based on a scale of 0–5 (Kitchenham 2007; Oxman 1994). The outcome and the corresponding scale are given in Fig. 12. Based on the testing outcome, the response was noted for each type of testing which is given below in Table 16.

In the Table 16 given above, the columns ‘GA based testing without predefined parameters-Response for Q1, GA based testing without predefined parameters -Response for Q2’ gives the response for question Q1 and Q2 for test subjects P1–P5 when applying GA base testing without pre-defined parameters and the column ‘GA based testing without pre-defined parameters-Sum of responses (Q1 + Q2)’ gives the added value of Q1 and Q2. Similarly, the columns ‘GA based testing with predefined parameters –Response for Q1, GA based test-

Fig. 12 Scale corresponding to difficulty level



ing with predefined parameters –Response for Q2’ gives the response for question Q1 and Q2 for test subjects P6–P10 when applying GA bases testing with appropriate pre-defined parameters and the column ‘GA based testing with predefined parameters -Sum of responses (Q1 + Q2)’ gives the added value of Q1 and Q2.

To perform the evaluation of GA based testing using appropriate pre-defined parameters and GA based testing without using pre-defined parameters, a Mann–Whitney U test is used. In order to calculate the U statistic, the next step is to rank the sum of the responses. The Table 17 gives rank for sum of responses for GA based testing using pre-defined parameters and GA based testing without using pre-defined parameters.

In Table 17 we can see that, ranks having tied score are given average value of ranks. Finally the actual ranks are noted. A Mann Whitney U test is applied for evaluating the ranks obtained. In Mann- Whitney U test, the U value is given as:

$$U = n_1 n_2 + n_x (n_x + 1) / 2 - T_x$$

Here n_1 and n_2 are the participants in each group and T_x is the largest of rank total and n_x is the number of participants in the group having largest rank total

In our example, $n_1 = 5$ and $n_2 = 5$, $T_x = 40$ and $n_x = 5$

Therefore $U = 5 * 5 + 5(5 + 1) / 2 - 40$

$U = 25 + 15 - 40$

$U = 0$

The next step is to find the critical value of U from the Mann–Whitney U test table of critical values (given in appendix). From the table at 5 % significance level for a two tailed test, the critical value of U for $n_1 = 5$ and for $n_2 = 5$ is 2.

Therefore $U = 2$ (from Mann Whitney U table at 5 % significance level)

To be statically significant, the obtained U value should be less than the critical value of U.

In our case $0 < 2$, which indicates that the difference we found during the testing of software with GA based method using appropriate pre-defined parameters and GA based methods without pre-defined parameters is unlikely to occur by chance. Testing the software with GA based method using appropriate pre-defined parameters and GA based method without pre-defined parameters makes a significant difference.

Therefore, Null Hypothesis (H_0) is rejected, which proves that, testing the software with GA based method using appropriate pre-defined parameters is more useful compared to testing the software with GA based method without using any pre-defined parameters.

Inference. It is always better to know the appropriate type and value of parameter setting to be used in GA based software testing.

Table 16 Response for the queries Q1 and Q2

Test subject	GA based testing without any pre-defined parameter settings-Response for Q1	GA based testing without any pre-defined parameter settings-Response for Q2	GA based testing without any pre-defined parameter settings-Sum of responses (Q1 + Q2)	Test subject	GA based testing with predefined parameters-Response for Q1	GA based testing with predefined parameters-Response for Q2	GA based testing with predefined parameters-Sum of responses (Q1 + Q2)
P1	5	4	10	P6	1	0	1
P2	5	4	9	P7	1	1	2
P3	5	5	10	P8	1	0	1
P4	5	4	9	P9	1	1	2
P5	5	5	10	P10	1	0	1

Table 17 Ranked response for GA based testing with and without using pre-defined parameters

Test subject	GA based testing without any pre-defined parameter settings-Sum of responses (Q1 + Q2)	Rank of the response	Actual rank of GA based testing without any pre-defined parameter settings	Test subject	GA based testing with predefined parameters-Sum of responses (Q1 + Q2)	Rank of the response	Actual Rank of GA based testing with predefined parameters
P1	10	8	9	P6	1	1	2
P2	9	6	6.5	P7	2	4	3.5
P3	10	9	9	P8	1	2	2
P4	9	7	9	P9	2	5	3.5
P5	10	10	6.5	P10	1	3	2
		Total	40			Total	13

5 Applications of this review paper

This section discusses the main applications of this paper. In the paper, we have made a review of GA based software testing and have given directions on how to improve quality of software testing using GA. As software has pervaded every aspect of human life, ensuring their reliability and correctness needs utmost importance. This confers a cardinal role for software testing in software development. We do not want the software to fail, as the repercussions that can occur in the event of a software malfunction can be huge, which may include loss of time, money and even life. All these can be prevented by ensuring the quality of the software. In practice, software testing still remains as the primary choice for assessing the quality of the software, as it can assess the validity and reliability of the software. Looking at the current trend in software testing, black box approaches are mostly used in industry. Though the black box approach assures some level of reliability for a software, it cannot be compared with the quality of test result given by structural testing. Though structural testing ensures reliability of the source code, the reluctance to use structural testing in practical software testing is due to the limited time, resources and large number of lines of code (LOC) which the testers have to handle during testing. In structural testing, exhaustive testing is not possible as it is not possible to test each and every combination of test data. Similarly manual testing of software is very tedious as it consumes a lot of time and effort. Meta-heuristic techniques such as GA may be used in such situations. In meta-heuristic techniques, the test data is generated automatically from a large search space. Though it may not be possible to find the accurate solution, an optimal or nearest solution may be found using GA.

From the review observations and experimental evaluations given in this paper, we found that while using GA for software testing the user should be aware of the best type of parameters and the optimum value of parameters. For a particular category of problem, if the user is able to apply GA based testing with appropriate parameter settings, it will be very helpful for the tester compared to manual testing. Therefore, if the tester is able to utilise the potential of GA in software testing based on the given guidelines provided in this paper, it will be a breakthrough in the field of software testing.

6 Limitations of GA based software testing

In the previous sections, we made a review on works which use GA for structural testing of software. Based on the review observations, we have also suggested some future research directions in software testing using GA. In this section we will discuss some the main disadvantages of GA based software testing. One of main issues of using GA in software testing is the design of fitness function. Designing a wrong or misleading fitness function will not cause convergence of the fitness function. This in turn hinders the process of test data generation. Another main disadvantage of GA based software testing is the execution time. Though the quality of solution obtained improves with the increase in time, it cannot be guaranteed to obtain an optimal solution in the specified time interval. Implementation difficulty is another issue in GA based software testing.

7 Threats to validity

The main threats to validity of this review may be due the literatures referred, method of conducting the review and from the observations made during the review.

7.1 Literature referred for review

The main threat to the validity of our work may be due to the limitation in the scope of the works which we have referred. We have limited our analysis to only those works which have mentioned the application of genetic algorithm in test data generation. For this, we have refined our search several times by the query “Genetic algorithm based software testing” and have finally identified a set of works which clearly falls within our area of focus. The downside of such restriction in the selection of works is that, all the possible variants of genetic algorithms and their applications may not have been analyzed which may have used different terms that don’t match our search query.

7.2 Method of conducting the review

We have made this study in order to expose the real practical issues in GA based software testing and to explore the research direction in GA based testing which has not received much attention till now. For this, we have made a study of the various parameters used in GA based testing. There exists a large number of parameters in GA and as the number of parameters increases, the combination of parameters also increases exponentially (Arcuri and Fraser 2011). The number of parameters also varies according to the system under test. Therefore, we have considered only the parameters which play a critical role in GA based testing. We feel that such a narrowing of the field of our study has sharpened its focus and enabled us to do an in depth analysis of our chosen study objectives; which being the identification of shortcomings of genetic algorithm and the future research scope in GA based software testing. We have not discussed any issues related to parameter control in this review. The value of parameters set initially need to be changed as the search based testing proceeds. We have not mentioned any issues in this area. Parameter control is still a non-discussed area in search based software testing, which has many issues to be explored, discussed and solved.

7.3 Observations made during the review

All the findings and suggestions made in this study have been made by analyzing the papers available in the literature carefully. We have conducted the review based on the facts drawn from the referred literature. Much more work and experiments may be done in order to substantiate the claims made in this work.

8 Conclusion

Of late, software test data generation has gained a wide interest among software engineers. In software testing, test data generation using search based techniques especially with meta-heuristic algorithms like GA have been widely explored by many researchers during the last decade. As large number of research works on test data generation using GA is being carried out, it is high time to conduct a study to throw light on the practical implications of this research trend. Therefore, we have tried to perform a systematic review to collect evidence from various literatures on the effectiveness of GA in software testing process. In this review, we have concentrated on the most critical aspects of GA based software testing: the effectiveness of GA in software testing and how to overcome the limitations of GA in practical software testing. Most of the works have tried to tide over the issues on population initialization, parameter settings and fitness function calculation as and when they arise without

addressing the basic cause. We feel that the present direction in GA based software testing is like attempting to treat symptoms. Treating symptoms will not cure sickness. For curing sickness, the disease itself needs to be treated. More holistic approaches and generalized solutions are the need of the hour in GA based testing. We have reviewed and suggested directions to tackle the root cause of these problems and correct it once and for all. For this, we have critically analyzed numerous works, studied pros and cons of various works and have set up some suggestions on how to improve the process of GA based testing. Therefore, we have put forth some suggestions for the researchers to set up some general guidelines on how to carry out the testing process, set up parameters and design fitness function in GA based software testing so that the potential of GA based software testing may be utilized to the full extent.

References

- Ahmed MA, Hermadi I (2008) GA-based multiple paths test data generator. *Comp Opera Res* 35:3107–3127
- Ali S, Briand LC, Hemmati H, Panesar-Walawege RK (2009) A systematic review of the application and empirical investigation of search-based test case generation. In: *IEEE Transactions on Software Engineering*, vol. 99
- Ali S, Iqbal MZ, Arcuri A, Briand L (2011) A search-based OCL constraint solver for model-based test data generation. In: *Proceedings of the 11th international conference on quality software*, pp. 41–50
- Alshahwan N, Harman M (2014) Coverage and fault detection of the output-uniqueness test selection criteria. In: *Proc. ISSTA*, pp. 1–12
- Arcuri A, Fraser G (2011) On parameter tuning in search based software engineering. In: *Proc. SSBSE*, pp. 33–47
- Arcuri A, Fraser G (2013) Parameter tuning or default values? An empirical investigation in search-based software engineering. *Empir Softw Eng* 18(3):594–623
- Beizer B (1990) *Software testing technique*. Van Nostrand Reinhold, New York
- Berndt DJ, Watkins A (2004) Investigating the performance of genetic algorithm-based software test case generation. In: *Proc. HASE*
- Binkley D, Harman M, Lakhotia K (2009) FlagRemove: a testability transformation for transforming loop assigned Flag. *ACM Trans Softw Eng Methodol* 2(3):110–146
- Bueno PM, Jino S (2002) Automatic test data generation for program paths using genetic algorithms. *Int J Softw Eng Knowl Eng* 12(6):691–709
- Campos J, Arcuri A, Fraser G, Abreu R (2014) Continuous test generation: enhancing continuous integration with automated test generation. In: *Proc. Automated Software Engineering (ASE)*
- Cao Y, Hu C, Li L (2009) An approach to generate software test data for a specific path automatically with genetic algorithm. In: *Proc. ICRMS*, Chengdu, pp. 888–892
- Chen C, Xu X, Chen Y, Li X, Guo D (2009) A new method of test data generation for branch coverage in software testing based on EPDG and genetic algorithm. In: *Proc. ASID*, pp. 307–310
- Doungsa-ard C, Dahal K, Hossain A, Suwannasart T (2007) Test data generation from UML state machine diagrams using gas. In: *Proc. ICSEA*
- Feldt R, Poulding S (2015) Broadening the search in search-based software testing: it need not be evolutionary. In: *Proc. Eighth international workshop on search-based software testing*, pp. 1–7
- Fischer M (2012) Generating test data for black-box testing using Genetic algorithms. In: *Proc. 17th ETFA*, pp. 1–6
- Fraser G, Arcuri A, McMinn P (2015) A memetic algorithm for whole test suite generation. *J Syst Softw* 103:311–327. doi:[10.1016/j.jss.2014.05.032](https://doi.org/10.1016/j.jss.2014.05.032)
- Fraser G, Arcuri A (2011a) EvoSuite: automatic test suite generation for object-oriented software. In: *Proc. ESEC/FSE*, Szeged, Hungary, pp. 416–419
- Fraser G, Arcuri A (2011b) It is not the length that matters, it is how you control it. In: *Proc. IEEE international conference on software testing, verification and validation*, pp. 150–159
- Fraser G, Arcuri A (2012) The seed is strong: seeding strategies in search-based software testing. In: *Proc. ICST*, pp. 121–130
- Fraser G, Arcuri A (2013) Whole test suite generation. *IEEE Trans Softw Eng* 39(2):276–291
- Fraser G, Arcuri A, McMinn P (2013) Test suite generation with memetic algorithms. In: *Proc. GECCO*

- Galeotti JP, Fraser G, Arcuri A (2014) Extending a search-based test generator with adaptive dynamic symbolic execution. In: Proc. ISSTA
- Girgis MR (2005) Automatic test data generation for data flow testing using a genetic algorithm. *J Univ Comp Sci* 11(5):898–915
- Goldberg D (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Boston
- Gonga D, Zhanga W, Yaob X (2011) Evolutionary generation of test data for many paths coverage based on grouping. *J Syst Softw* 84:2222–2233
- Graham-Rowe D (2002) Radio emerges from the electronic soup. *New Sci* 175(2358):19
- Harman M (2007) The current state and future of search based software engineering. In: Proc. FOSE, pp. 342–357
- Harman M, Mansouri SA, Zhang Y (2012) Search-based software engineering: trends, techniques and applications. *ACM Comp Surv* 45(1):1–66
- Harman M, Jia Y, Zhang Y (2015) Achievements, open problems and challenges for search based software testing. In: Proc. 8th IEEE international conference on software testing, verification and validation
- Harman M, McMinn P (2010) A theoretical and empirical study of search based testing: local, global and hybrid search. *IEEE Trans Softw Eng* 36(2):226–247
- Harman M, McMinn P, Wegener J (2007) The impact of input domain reduction on search based test data generation. In: Proc. ESEC/FSE, Croatia, pp. 155–164
- Hermadi I, Lokan C, Sarker R (2014) Dynamic stopping criteria for search-based test data generation for path testing. *Inf Softw Technol J* 56(4):395–407
- Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor
- Jones B, Sthamer H, Eyres D (1996) Automatic structural testing using genetic algorithms. *Softw Eng J* 11(5):299–306
- Jorgensen PC (2008) Software testing: a Craftsman's approach. Auerbach Publications, Boca Raton
- Khor S, Grogono P (2004) Using a genetic algorithm and formal concept analysis to generate branch coverage test data automatically. In: Proc. ASE
- Kitchenham BA (2007) Guidelines for performing systematic literature reviews in software engineering. Technical report EBSE- 2007-01
- Korel B (1990) Automated software test data generation. *IEEE Trans Softw Eng* 16(8):870–879
- Latiu GI, Cret OA, Vacariu L (2012) Automatic test data generation for software path testing using evolutionary algorithms. In: Proc. Third international conference on emerging intelligent data and web technologies, pp.1–8
- Li J, Baob W, Zhaoa Y, Maa Z, Donga H (2009) Evolutionary generation of unique input/output sequences for class behavioral testing. *Comput Math Appl* 57:1800–1807
- Lin P, Bao XL, Shu ZY, Wang XJ, Liu JM (2012) Test case generation based on adaptive genetic algorithm. In: International conference on quality, reliability, risk, maintenance, and safety engineering, Chengdu, China, pp. 863–866
- Liu D, Wang X, Wang J (2013) Automatic test case generation based on genetic algorithm. *J Theor Appl Inf Technol* 48(1):411–416
- Louzada J, Camilo-Junior CG, Vincenzi A, Rodrigues C (2012) An elitist evolutionary algorithm for automatically generating test data. In: Proc. IEEE WCCI, Brisbane, Australia
- Malburg J, Fraser G (2011) Combining search based and constraint-based testing. In: Proc. IEEE ASE, Lawrence, KS, pp. 436–439
- Mantere T, Alander JT (2005) Evolutionary software engineering, A review. *J Appl Soft Comput* 5:315–333
- Mao C, Yu X (2013) Test data generation for software testing based on quantum-inspired genetic algorithm. *Int J Comput Intell Appl* 12(1) pp. 1350004 (21 pages)
- McMinn P (2004) Search-based software test data generation: a survey. *Softw Test Verif Reliab* 14(2):105–156
- McMinn P (2005) Evolutionary search for test data in the presence of state behavior. Ph. D. Thesis, University of Sheffield, Sheffield
- McMinn P (2011) Search-based software testing: past, present and future. In: Proc. 4th international conference on software testing, verification and validation workshops, pp. 153–163
- McMinn P (2013) An identification of program factors that impact crossover performance in evolutionary test input generation for the branch coverage of C programs. *Inf Softw Technol* 55:153–172
- Michael CC, McGraw GE, Schatz MA (2001) Generating software test data by evolution. *IEEE Trans Softw Eng* 27(12):1085–1110
- Miller J, Reformat M, Zhang H (2006) Automatic test data generation using genetic algorithm and program dependence graphs. *Inf Softw Technol* 48:586–605
- Myers G (1979) The art of software testing. Wiley, New York
- Oxman AD (1994) Systematic reviews: checklists for review articles. *BMJ* 309(6955):648–651

- Pachauri A, Gursaran (2011) Software test data generation using path prefix strategy and genetic algorithm. In: Proc. international conference on science and engineering, pp. 131–140
- Pargas RP, Harrold MJ, Peck RR (1999) Test data generation using genetic algorithms. *J Softw Test Verif Reliab* 9:263–282
- Pei M, Goodman ED, Gao Z, Zhong K (1994) Automated software test data generation using a genetic algorithm. Technical report, Michigan State University
- Pocatiu P, Ivan I (2013) A genetic algorithm based system for automatic control of test data generation. *Stud Inf Control* 22(2):219–226
- Rauf A, Anwar S, Jaffer MA, Shahid A (2010) Automated GUI test coverage analysis using GA. In: Proc. 7th international conference on information technology, pp. 1057–1062
- Roper M, Maclean I, Brooks A, Miller J, Wood M (1995) Genetic Algorithms and the automatic generation of test data. Technical report RR/95/195[EFoCS-19-95], Department of Computer Science, University of Strathclyde
- Shamshiri S, Rojas JM, Fraser G, McMinn P (2015) Random or genetic algorithm search for object-oriented test suite generation?. In: Proc. Genetic and Evolutionary Computation Conference (GECCO 2015), pp. 1367–1374
- Sharma N, Pasala A, Kommineni R (2012) Generation of character test input data using GA for functional testing. In: Proc. 19th APSEC-SATA workshop, pp. 87–94
- Sofokleous AA, Andreou AS (2008) Automatic, evolutionary test data generation for dynamic software testing. *J Sys Softw* 81:1883–1898
- Srinivas M (1994) Genetic algorithms: a survey. *J Comput* 27:17–26
- Sthamer HH (1996) The automatic generation of software test data using genetic algorithms. Ph. D thesis, University of Glamorgan, Pontypridd, Wales
- Suresh Y, Rath SK (2013a) A genetic algorithm based approach for test data generation in basis path testing. In: International Journal of Soft Computing and Software Engineering [JSCSE], Special Issue: The proceeding of international conference on soft computing and software engineering, 3(3)
- Suresh Y, Rath S (2013b) Application of meta-heuristic algorithms for automated software test data generation. *Int J Comput Intell Stud* 4(2):113–133
- Tracey N (2000) A search-based automated test data generation framework for safety critical software. Ph. D. thesis, University of York
- Watkins A (1995) The automatic generation of test data using genetic algorithms. In: Proc. 4th software quality conference, pp. 300–309
- Wegener J, Baresel A, Sthamer H (2001) Evolutionary test environment for automatic structural testing. *J Inf Softw Technol* 43:841–854
- Xanthakis S, Ellis C, Skourlas C, Le Gall A, Katsikas S, Karapoulis K (1992) Application of genetic algorithms to software testing. In: Proc. 5th ICSE, pp. 625–636
- Xiao J, Afzal W (2010) Search-based resource scheduling for bug fixing tasks. In: Proc. 2nd SSBSE, pp. 133–14
- Xue-ying MA, Sheng BK, Zhen-feng HE, Cheng-qing YE (2005) A genetic algorithm for test-suite reduction. In: Proc. ICSMC, pp. 133–139
- Zhang N, Wu B, Bao X (2015) Automatic generation of test cases based on multi-population genetic algorithm. *Int J Multimed Ubiquitous Eng* 10(6):113–122
- Zhang W, Gong D, Yao X, Zhang Y (2007) Evolutionary generation of test data for many paths coverage. In: Proc. Chinese Control and Decision Conference, pp. 230–235

Reproduced with permission of
copyright owner. Further
reproduction prohibited without
permission.