Heuristic search-based approach for automated test data generation: A survey

Article *in* International Journal of Bio-Inspired Computation · January 2013

DOI: 10.1504/IJBIC.2013.053045

CITATIONS
13

READS
459

2 authors:

Ruchika Malhotra
University of Information Technology
65 PUBLICATIONS **1,145** CITATIONS
SEE PROFILE

Manju Khari
Ambedkar Institute of Advanced Communication Technologies and Research
51 PUBLICATIONS **111** CITATIONS
SEE PROFILE

Heuristic search-based approach for automated test data generation: a survey

Ruchika Malhotra

Department of Software Engineering,
Delhi Technological University,
Bawana Road, Delhi 110042, India
E-mail: ruchikamalhotra2004@yahoo.com

Manju Khari*

Department of Computer Science and Engineering,
Ambedkar Institute of Advanced Communication
Technology and Research,
Geeta Colony, Delhi 110032, India
E-mail: manjukhari@ymail.com
*Corresponding author

Abstract: The complexity of software has been increasing in the past few years, and software testing as a most intensive factor is becoming more and more expensive. Testing costs often account for up to 50% of the total expense of software project development; hence any techniques leading to the automatic test data generation will have great potential to substantially reduce these costs. Existing approaches of automatic test data generation have achieved some success by using heuristic search-based approach, but they are not summarised. In this paper we presented a survey on heuristic search-based approach, i.e., genetic algorithm for automated test data generation. We summarise the work done by researchers those who have applied the concept of heuristic search-based approach for test data generation. The main objective of this paper is to acquire the concepts related to heuristic search-based approach for automated test data generation and moreover the use of heuristic search-based approach fascinated by the fact that many of the testing problems can be formulated as search problem. We also provide constructive guidelines for future research for automated test data generation.

Keywords: genetic algorithm; heuristic search; automated test data generation.

Reference to this paper should be made as follows: Malhotra, R. and Khari, M. (2013) 'Heuristic search-based approach for automated test data generation: a survey', *Int. J. Bio-Inspired Computation*, Vol. 5, No. 1, pp.1–18.

Biographical notes: Ruchika Malhotra is an Assistant Professor at the Department of Software Engineering, Delhi Technological University, Delhi, India. She was an Assistant Professor at the University School of Information Technology, Guru Gobind Singh Indraprastha University, Delhi, India. Prior to joining the school, she worked as a full-time research scholar and she received her Doctoral research fellowship from the University School of Information Technology, Guru Gobind Singh Indraprastha Delhi, India. She received her Masters and Doctorate degree in Software Engineering from the University School of Information Technology, Guru Gobind Singh Indraprastha University, Delhi, India. She is a co-author of book titled *Object Oriented Software Engineering*. Her research interests are in software testing, improving software quality, statistical and adaptive prediction models, software metrics, neural nets modelling, and the definition and validation of software metrics. She has published more than 65 research papers in international journals and conferences.

Manju Khari is a research scholar with Delhi Technological University (formerly Delhi College of Engineering), Delhi, India. She was an Assistant Professor in Ambedkar Institute of Advanced Communication Technology and Research, Guru Gobind Singh Indraprastha University, Delhi, India. She received her Masters degree in Information Security from Ambedkar Institute of Advanced Communication Technology and Research, Guru Gobind Singh Indraprastha University, Delhi, India. Her research interests are software testing, software quality, software metrics and genetic algorithm. She has published several papers in international journals and conferences.

1 Introduction

Software testing is the most substantial analytical quality assurance measure for software systems. Often, more than 40% to 50% of the entire development expenditure is allocated to software testing. The crucial activity for the test quality is the test data generation, since type and scope of the test data are determined by selecting feasible test cases. Existing test data generation methods can essentially be differentiated into black box tests and white-box tests. In black box testing, test cases are determined from the specification of the programme under test, whereas in the white-box testing, they are derived from the internal structure. In both cases, automated data generation is difficult. Automation of the black-box test is only meaningfully possible if a formal specification exists. A wide range of commercial tools is available to support white-box tests, e.g., TESSY (http://www.ats-software.de/html/prod_tessy.htm), Cantata (http://www.iplbath.com/p4.htm), Attol Unit Test (http//www.rational.com/products/testrt/index.jsp) and TCAT (http://www.soft.com/TestWorks). However, these tools are limited to programme code instrumentation and coverage measurement. The test case design itself is further reliant on the tester. Heuristic search-based approach presents both a flexible and a powerful dynamic search method they are working properly to finding good solution in a large and complex search spaces.

A survey brings additional benefits as compared to a simple study because it not only simplifies evaluation at the same time it renders of all available research work relevant to a particular phenomenon of interest (Mahmood, 2007). The study originates from legitimate motive that the techniques that are discussed in past reflect their suitability for the present scenario and are considered to fulfil the needs of the future perceptivity.

Developments in the field of 'automated test data generation' were initiated in early 70s. Articles like 'Testing large software with automated software evaluation systems' by Ramamoorthy and Ho, in 1975 and 'Automatic generation of floating-point test data' by Miller and Spooner, in 1976, are a few examples of the early work in this field (Hannay et al., 2007). Nevertheless, Clarke's paper of 1976 is considered to be the first of its kind to proposed an algorithm for automated test data generation which is written in FORTRAN. Since 1970 researchers were working on test data generation, but regrettably current status of software world has hardly any fully automated test data generation working tool available in industry. In the beginning from 70s to mid of 80s researchers were working on test data generation with using symbolic execution and language used for implementation was based on FORTRAN (Hennel et al., 1989; Clarke, 1989; McCabe, 1989). In mid 80s itself, Prather and Myers (1989) introduced a new concept, i.e., path prefix for the same. Korel et al. (1996) had made lot many changes and test data generation moves towards object-oriented approaches and finally in 20s lost many authors work on hybrid testing methods, detect infeasible paths for saving computational time (Ali and Briand, 2010). In 2010, Tahbildar et al. proposed a heuristic approach for find out the number of iteration required for longest path coverage in 2000–2010 object-oriented test data generation approaches are chosen as core area of research. Industry is using object-oriented approaches as well as techniques for software development because of high productivity. UML has got a great height for software testing of object-oriented approaches. Mutation testing technique is used to improve reliability of object-oriented software with a scalable data generation based on multi agents (Gong et al., 2011; Offutt et al., 1995). Ma et al. (2007) has proposed a technique for test data generation even if more or less path predicate is unsolvable. But the proposed method could not provide a good coverage. Tahbildar and Kalita (2010) proposed a heuristic approach useful for construct having loops of different dimensions and array of variable length.

As far as objectives are concerned, in this study we are going to concentrate on the heuristic search-based approach for automated test data generation which is a part of software testing. Software is further divided in three categories they are functional testing, structural testing and grey hybrid testing (Jones et al., 1996). These categories are further sub divided in other categories. Automated test data generation is a part of all categories since we have to generate test data for testing in each and every category. The purpose of this survey is to collect and assess the studies on heuristic search-based approach for automated test data generation.

The rest of the is organised as follows: in Section 2 we discussed about the selection criteria of the specified topic, i.e., heuristic search-based approach for automated test data generation and it include sub topic named as selection of search engines, searching of required material and storage criterion of searching material. In Section 3 we mentioned few research issues related to our survey. In this section basically describe some research question which comes to our mind while conducting survey. These research questions are mentioned with a short solution because their description is defined in other section in brief. In Section 4 we discuss the basic terms carried out in our research background. What actually the automated test data generation is? And what is genetic algorithm (GA) how it defined all are mention in this above said section. After that in Section 5 we mention the complete literature study which is compiled with the help of past researchers work and we also provide a summarised table for the same. After that we mentioned the analysis of current trends followed by Section 6. Finally at the end we mentioned the future scope and few challenges with respect to heuristic search-based approach for automated test data generation continued with the references list.

2 Selection criteria

This section explains the research procedure adopted for identification of studies in the literature using heuristic search-based approach for automated test data generation

techniques. The main aim of the selection criteria is to extract the relevant and significant research papers based on heuristic search-based approach for automated test data generation. To continue this selection criterion we have to filter our relevant data from the huge pool of available information on software testing technique, we have to focus only on automated and semi automated test data generation techniques using heuristic search-based approach. Figure 1 depicts the actual development of the selection criteria. It is basically a pre selection process for the designs relevant to the formulation for search under this title.

The selection process relies upon famous and authentic search engines, journals, conferences and articles. Hence we have to explore rich database, checking out the lists of sources and assuring that all relevant research papers were covered was considered to be a good set of measures for the establishment of the selection criteria. Figure 2 depicts a step by step process of research method. The selection criteria can be sub divided into three steps and these steps are summarised below (Kitchenham, 2004):

1 selection of the search engines, articles, conference proceedings and journals

2 search a required material and their after contributing advancement in the search results

3 storage criterion of the searched material.

2.1 Selection of search engines, articles, conference proceedings and journals

Initial stage for searching and for achieving a desired goal in a minimum number of steps, it is mandatory to choose appropriate search engines. We have to consider the following points to support the selection criterion of search engine (Kitchenham, 2004):

1 search engines are renowned and authentic for providing scholarly material so that we extract the relevant and expected search results

2 search engines are targeted and can lead straight to the required research material

3 search engine provide advanced search facilities like refinement of material up to different levels.

List of the selected search engines:

1 ACM Digital Library

2 IEEE explore

3 Springer

4 Science direct

5 Compendex

6 Google scholar.

Figure 1 Development of selection criteria

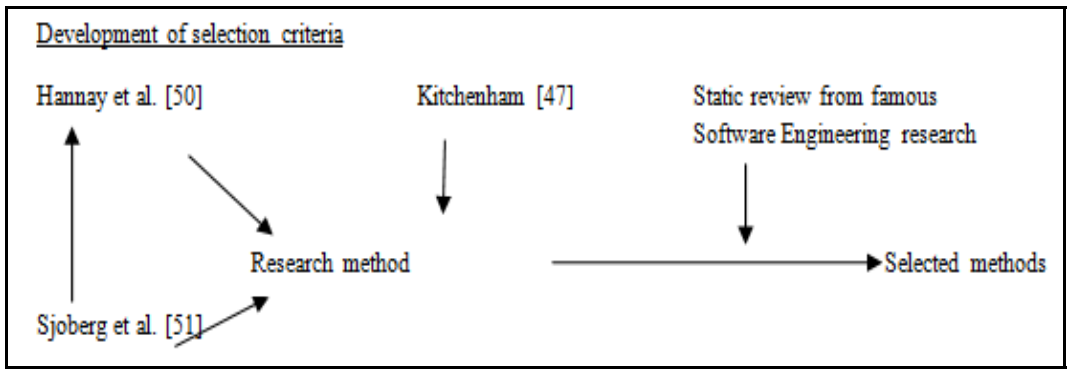
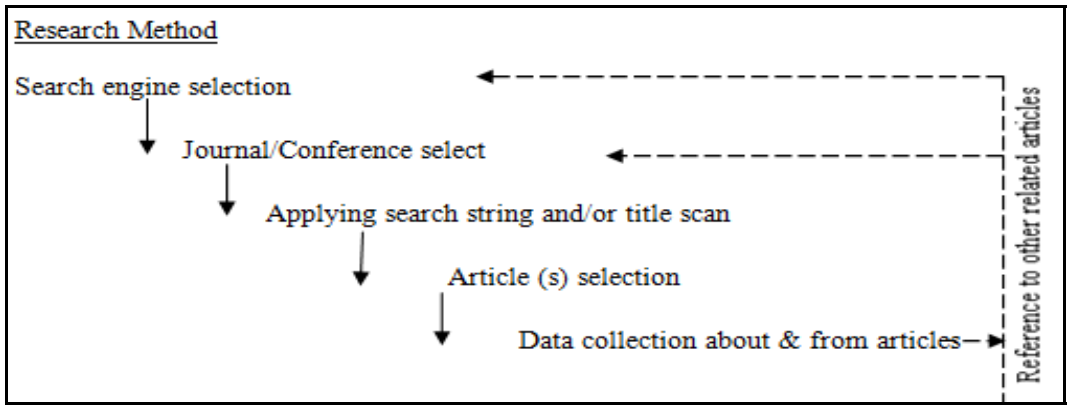


Figure 2 Process of research method



List of selected journals/ conferences are given below:

1 *Software Testing, Verification and Reliability* (*STVR*)

2 *Automated Software Engineering Journal* (*ASEJ*)

3 *Empirical Software Engineering Journal* (*EMSE*)

4 *ACM Transactions on Software Engineering Methodology* (*TOSEM*)

5 *IEEE Transactions on Software Engineering* (*TSE*)

6 *IEE/IEEE Software* (*ISW*)

7 International Conference on Software Engineering (ICSE)

8 European Software Engineering Conference/ Foundation of Software Engineering (ESEC/FSE)

9 International Conference on Automated Software Engineering (ASE).

To attain high search rate of the relevant articles it was decided that a broad research area base of well known journals and conference proceedings would be set up for search process. Moreover only the relevant and scholarly search engines are to be kept on highest priority and to be considered for the search. It was also required that only the accomplished key words or string would be used for exploring. For search any keyword we have to apply this principal, i.e., 'From more specific to general' schema would be adopted because it was qualified to rapidly highlight the required targeted articles.

List of keywords preferably used for searching:

1 automatic test data generation

2 test data generation

3 testing technique

4 heuristic search

5 search-based approaches

6 test case generation.

2.2 Searching of required material

This section is useful for searching required material for extracting the relevant material according to our research topic. Searching of required material consist mainly two steps they are as follows:

1 for searching any journal or conference we have to search through year to year with the help of searching keyword

2 searched materials were improved on the foundation of the relevant material that had been collected during detailed examination of the selected material.

2.3 Storage criterion of searched material

Since properly saving of the searched material is as important as the search process itself. Lot many procedures were adopted to store the collected material according to the individual requirement and comfort. Storage procedures were followed with a few idea involved in it they are: wrong storage of searched or selected material, leaving few searched or selected material without storage, duplication of material in the database.

A small database was created for storing the searched or selected material. It would be in Microsoft Word or in Excel one table contain data about searched or selected material including the categories of title name, title year, name of publisher/journal/conference, name of the authors and key terms. The database also establishes preliminary category of searched or selected material and brief description of the data which were talked about in searched or selected material. This kind of database is modelled to store full record of a searched or selected material by giving it a storage number along with a cross reference. This procedure blocks duplicate entry of a searched or selected material. All this is to be done with one principal that no material was save without making and entry in the database.

2.4 Graphical analysis of searched or selected material

An analysis of the searched or selected material indicates that the research study or the trends of publishing the research study was not evenly distributed over the years of the defined period, i.e., 1989–2011. Figure 3 depicts a total collection of research papers and Table 1 defined a list of selected reference papers. These total collection of selected research paper are divided in three parts they are No. of journals, No. of conferences and No. of detailed study material, detailed study material include one or two technical reports, books and few documentation of tools. In fact this study showed a lot of variation, as shown in the bar graph mentioned below. In this graph, year-wise publication of research papers on heuristic searched-based test data generation vs. calendar year is shown. It serves the purpose to analyse the published searched or selected material/data in different years. Figure 4 depicts a bar graph representation of distribution of searched or selected material research papers on heuristic search-based automated test data generation techniques published in selected journals and conference proceedings of the decade 1989–2011.

Table 1 List of selected research paper

Selected research journal/conference	*No. of papers*
Journals	39
Conferences	11
Detailed material	8
Total	58

Figure 3 Number of research papers selected (see online version for colours)

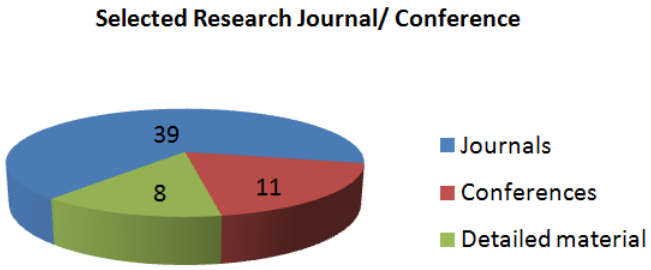
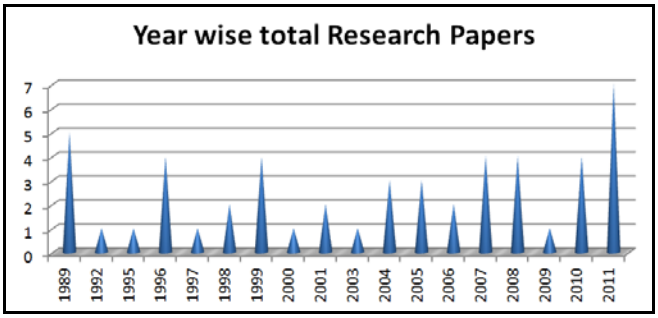


Figure 4 Year wise distribution of selected research material (see online version for colours)



As a result analysis of the selection step 58 research papers were selected for detail examination out of the total evaluations of 79 research papers. However, due to irrelevancy of title 21 of these research papers were removed from the selected list. Finally, 58 research papers were selected for further proceedings. These research papers were only related to the Automated/semi automated test data generation techniques based on heuristic approach and pertained to the selected journals or conference proceedings for the period of 1989–2011.

3 Research issues

The survey is conducted with the aim of identifying some research questions. They are usually performed on the basis of the existing evidence for a particular topic. We summarised these evidence and aid in the identification of gap in the current research and thus can form the basis of new research activity. We addressed some research question they are mentioned below with their solution:

RQ. 1 Which heuristic search-based approaches have been used for automated test data generation?

RQ. 2 Which type of automated test data has been generated?

RQ. 3 For which testing level heuristic search-based testing approach have been used more frequently?

RQ. 4 For which testing strategies has heuristic search-based approach have been used more frequently?

RQ. 5 What are the most used alternatives to which heuristic search-based approach for automated test?

RQ. 6 Which automated test data generation testing field is used more frequently?

RQ. 7 Which automated test data generation testing approaches is used more frequently?

RQ. 8 Which automated test data generation Implementation technique is used more frequently?

4 Research background

In this section we discussed about the basic aspect of our survey to be carried out which include the concepts of automated test data generation and GA in brief. These concepts are again sub divided in defined below.

4.1 Automated test data generation

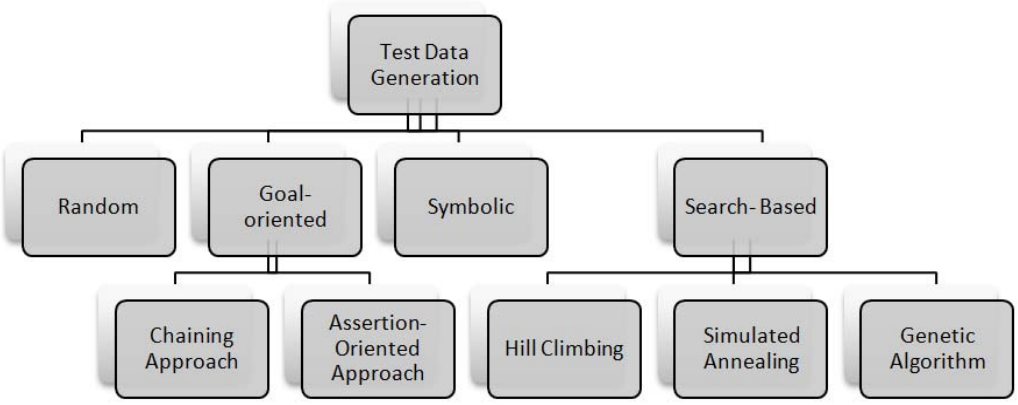
Test data generation, an important part of software testing, is the process of creating a set of data for testing the adequacy of new or revised software applications. Good quality test data is the key to the success of new and existing applications. Being able to generate test data quickly for a range of test cases will save organisations time and money. Testing is often performed on copies of production databases, which can lead to loss of productivity during the testing process and exposure of sensitive live data. The generated data is also enriched with coverage techniques to ensure all combinations of data exist automatically.

It involves two things:

- select some testing criteria
- generate test cases that meet the criteria.

A test data generation technique should be able to distinguish between good and bad test cases. Figure 5 depicts various methods of test data generation, they are random test data generation, goal-oriented test data generation, symbolic test data generation and search-based test data generation. There has been a significant evolution in the methods of test data generation which are described in detailed below.

Figure 5 Various methods of test data generation



4.1.1 Random test data generation

Random testing is a form of functional testing that is useful when the time needed to write and run directed tests is too long (or the complexity of the problem makes it impossible to test every combination). Release criteria may include a statement about the amount of random testing that is required. One of the big issues of random testing is to know when a test fails. In this testing, test case values from the input domain of the programme are selected and are provided to the programme under test. This type of testing is relatively cheap method of generating initial test data.

4.1.2 Goal-oriented test data generation

The goal-oriented approach provides guidance towards a certain set of paths. The test data generators in this approach generate an input for any path u instead of the usual approach of generating input from the entry to the exit of a block of code. Thus, the generator can find any input for any path p which is a subset of the path u. This drastically reduces the risk of generating relatively infeasible paths and provides a way to direct the search. Two methods follow this technique:

- The chaining approach

  The chaining approach tries to identify a chain of nodes that are vital to the execution of the goal node. The chaining approach starts by executing for any arbitrary input x. The search programme, during the execution of each branch decides whether to continuation of execution through this branch or if an alternative branch be taken because the current branch does not lead to the goal node (Ferguson and Korel, 1996; Prather and Myers, 1989).

- Assertion-oriented approach

  The assertion-oriented approach is an extension of the chaining approach. In this approach assertions – that is constraint conditions are inserted. This can be done either manually or automatically. If the programme does not hold on execution there is an error in the programme or the assertion (Korel and Al-Yami, 1996).

  When an assertion is executed it must hold, otherwise there is an error either in the programme or in the assertion.

4.1.3 Symbolic test data generation

Symbolic execution refers to the analysis of programmes by tracking symbolic rather than actual values, a case of abstract interpretation. The field of symbolic simulation applies the same concept to hardware. Symbolic computation applies the concept to the analysis of mathematical expressions.

If the symbolic expression can be solved the test path is feasible and the solution corresponds to a set of input data which will execute the test path. If no solution can be found then the test path is infeasible.

Common problem that occurs is that manipulating algebraic expressions is expensive when performed on a large number of paths.

4.1.4 Search-based test data generation

Local search is a meta heuristic method for solving computationally hard optimisation problems. Local search can be used on problems that can be formulated as finding a solution maximising a criterion among a number of candidate solutions. Local search algorithms move from solution to solution in the space of candidate solutions (the search space) by applying local changes, until a solution deemed optimal is found or a time bound is elapsed. This technique is likely to trap in local minima or maxima, thus providing test case values that are not globally optimal (Kanmani and Maragathavalli, 2010b).

In contrast to local search-based testing techniques, heuristic search testing techniques are based on using the heuristic search techniques to generate test data. Later one is an approximation of optimal search technique. The use of these techniques in testing is based on the fact that the testing problems can be formulated as the problems of searching through the input domain of the programme those input values that satisfy predefined testing criteria (McMinn, 2004).

Meta-heuristics are commonly used for combinatorial optimisation, where the search space can become especially large (Bueno et al., 2011b). Many practically important problems are NP-hard. Heuristic search algorithms handle an optimisation problem as a task of finding a 'good enough' solution among all possible solutions to a given problem, while meta-heuristic algorithms are able to solve even the general class of problems behind the certain problem.

In order to use search algorithms in software engineering the first step is that the particular software engineering problem should be defined as a search problem. After this has been done, a suitable algorithm can be selected and the issues regarding that algorithm must be dealt with. There are three common issues that need to be dealt with by any search algorithm:

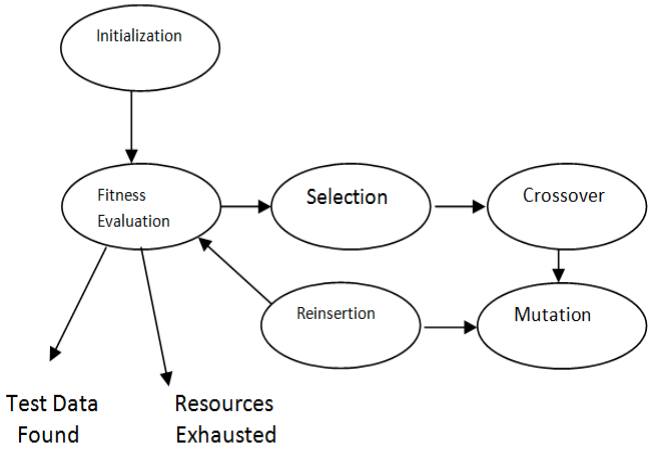
- encoding the solution
- defining transformation
- measuring the goodness of a solution.

To understand the basic concepts behind the approaches presented here, the most commonly used search algorithms are briefly introduced. The most common approach is to use GAs. Hill climbing and its variations, is also quite popular due to its simplicity. Finally several studies use simulated annealing.

- *Hill climbing:* Hill climbing starts at a random point in the search space. Points in the search space neighbouring the current point are evaluated for fitness. If a better candidate solution is found, hill climbing moves to that new point, and evaluates the neighbourhood of that candidate solution. This step is repeated, until the neighbourhood of the current point in the search space offers no better candidate solutions; a so-called 'local optima' (Harman and McMinn, 2007). If the local optimum is not the global optimum the search may benefit from being 'restarted' and performing a climb from a new initial position in the landscape.
- *Simulated annealing:* search by simulated annealing is similar to hill climbing, except movement around the search space is less restricted. Moves may be made to points of lower fitness in the search space, with the aim of escaping local optima. This is dictated by a probability value that is dependent on a parameter called the 'temperature', which decreases in value as the search progresses. The lower the temperature, the less likely the chances of moving to a poorer position in the search space, until 'freezing point' is reached, from which point the algorithm behaves identically to hill climbing. simulated annealing is named so because it was inspired by the physical process of annealing in materials.
- *GAs:* GAs are a form of 'global' search, sampling many points in the search space at once. GAs are inspired by Darwinian evolution and the concept of survival of the fittest.

Each point in the search space currently under consideration is referred to as an 'individual' or a 'chromosome'. The current set of individuals currently under consideration is collectively referred to as the current 'population'. The first population is randomly generated, and each individual is evaluated for fitness. A selection mechanism, biased towards the best individuals, decides which individuals should be parents for crossover. During crossover, elements of each individual are recombined to form two offspring individuals that embody characteristics of their parents. Subsequently, elements of the newly-created chromosomes are mutated at random, with the aim of diversifying the search into new areas of the search space. Figure 6 depicts the basic steps of GA which includes initialisation, fitness function, selection, crossover, mutation and re-insertion for generation of test data.

Figure 6 Overview of the main steps of a GA



In general, there are two requirements that need to be fulfilled in order to apply a search-based optimisation technique to a testing problem:

1 *Representation*. The candidate solutions for the problem at hand must be capable of being encoded so that they can be manipulated by the search algorithm – usually as sequences of elements as for chromosomes with a GA.

2 *Fitness function*. The fitness function guides the search to promising areas of the search space by evaluating candidate solutions. The fitness function is problem-specific, and needs to be defined for a new problem. The next section discusses some fitness functions that have been used by different authors in search-based software testing.

4.2 Genetic algorithm

4.2.1 Introduction to GAs

GAs are the heuristic (stochastic) search algorithms that are used to solve a variety of optimisation problems (Goldberg, 1989). GAs are used in software testing to facilitate the optimisation problem.

It mimics the process of natural biological evolution and the Darwin's principle of survival of the fittest.

In essence, GAs are a computer model of biological evolution. An important characteristic of GAs is the fact that they are very effective when searching or optimising spaces that is not smooth or continuous.

GA's are iterative procedures that produce new populations at each step. A new population is created from an existing population by means of performance evaluation, selection procedures, recombination and survival. These processes repeat themselves until the population locates an optimum solution or some other stopping condition is reached.

GAs differentiated from other conventional techniques due to:

1 GA a representation for the sample population must be derived.

2 GAs manipulates directly the encoded representation of variables, rather than manipulation of the Variables themselves.

3 GAs use stochastic rather than deterministic operators.

4 GAs search blindly by sampling and ignoring all information except the outcome of the sample.

5 GAs search from a population of points rather than from a single point; thus reducing the probability of being stuck at a local optimum, which make them suitable for parallel processing. In the context of software testing, the basic idea is to search the domain for input variables, which satisfy the goal of testing.

GAs find application in bioinformatics, phylogenetics, computational science, engineering, economics, chemistry, manufacturing, mathematics, physics and other fields.

A typical GA requires:

1 a genetic representation of the solution domain

2 a fitness function to evaluate the solution domain.

A standard representation of the solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary.

The fitness function is defined over the genetic representation and measures the quality of the represented solution. The fitness function is always problem dependent (Claudia and Regina, 2003).

Once the genetic representation and the fitness function are defined, a GA proceeds to initialise a population of solutions and then to improve it through repetitive application of the mutation, crossover, inversion and selection operators.

4.2.2 Basics of GA

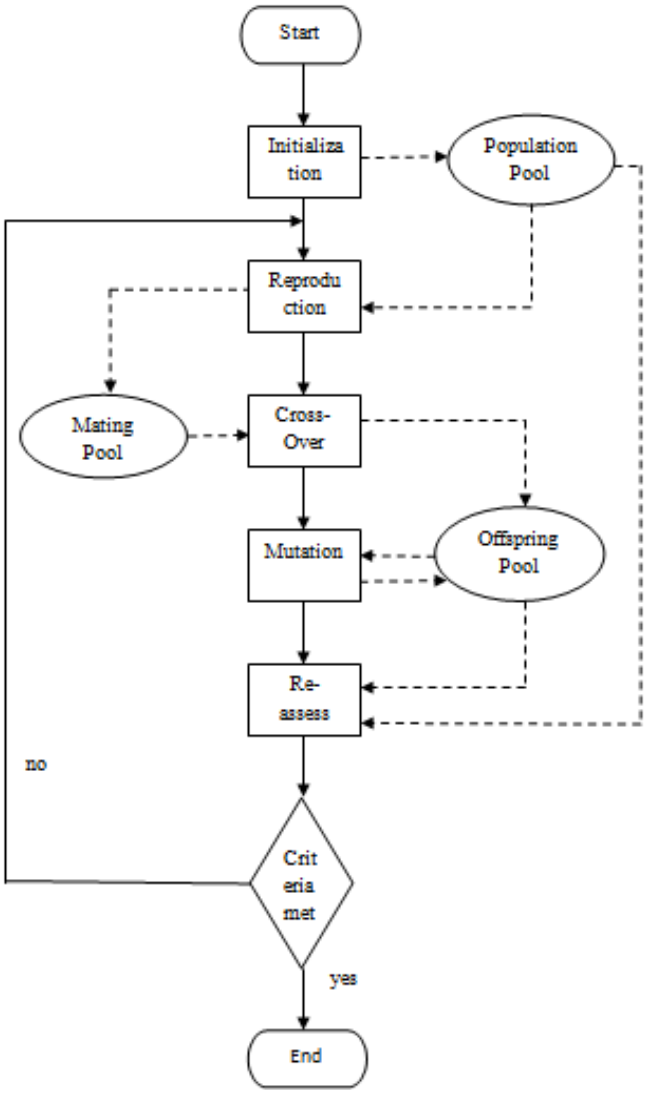
GA maintains a population pool of candidate solutions called strings or chromosomes. Each chromosome is a collection of building blocks known as genes, which are instantiated with values from finite domain.

Let p, q denote the value of gene q in chromosome p in the population. Associated with each chromosome is a fitness value which is determined by a user determined function, called the fitness function. The function returns the magnitude that is proportional to the candidate solutions suitably and/or optimality. At the start of the algorithm, an initial population is generated. Initial members of the population may be randomly generated, or generated according to some rules. The reproduction operator selects chromosomes from the population to be parents for a new chromosome and enters them into the mating pool. Selection of a chromosome for parenthood can range from a totally random process to one that is biased by the chromosome's fitness. The cross-over operator oversees the mating process of two chromosomes. Two parent chromosomes are selected from the mating pool randomly and the cross-over rate, which is a real number between zero and one, determines the probability of producing a new chromosome from the parents. If the mating was performed, a child chromosome is created which inherits complementing genetic material from its parents. The cross-over operator decides what genetic material from each parent is passed onto the child chromosome. The new chromosome produced is entered into the offspring pool. This new chromosome may represent an unexplored point in the search space. The mutation operator takes each chromosome in the offspring pool and randomly change part of its genetic make-up, i.e., its content. The probability of mutation occurring on any chromosome is determined by the user specified mutation rate. Chromosomes mutated or otherwise, are put back into the offspring pool after the mutation process. Thus each new generation of chromosomes are formed by the action of genetic operators (reproduction, cross-over and mutation) on the older population. Finally, the members of the population pool are compared with those of the offspring pool. The chromosomes are compared via their fitness value to derive a new population, where the weaker chromosomes may be eliminated. In exact, weaker members in the population pool are replaced by the better child chromosomes from the offspring pool. The heuristic for assessing the survival of each chromosome into the next generation is called the replacement strategy. The process of reproduction,

cross-over, mutation and formation of a new pollution completes one generation cycle. A GA is left to progress through generations, until certain criteria (such as a fixed number of generations, or a time limit) are met.

GAs were initially used for machine learning systems, but it was soon realised that GAs have great potential in function optimisation. Let p, q denote the value of gene q in chromosome p in the population. Associated with each chromosome is a fitness value which is determined by a user determined function, called the fitness function. The function returns a magnitude that is proportional to the candidate solution's suitably and/or optimality. Figure 7 shows the control and data flow of a canonical GA. At the start of the algorithm, an initial population is generated. Initial members of the population may be randomly generated, or generated according to some rules. The reproduction operator selects chromosomes from the population to be parents for a new chromosome and enters them into the mating pool. Selection of a chromosome for parenthood can range from a totally random process to one that is biased by the chromosome's fitness. The cross-over operator oversees the mating process of two chromosomes. Two parent chromosomes are selected from the mating pool randomly and the cross-over rate, which is a real number between zero and one, determines the probability of producing a new chromosome from the parents. If the mating was performed, a child chromosome is created which inherits complementing genetic material from its parents. The cross-over operator decides what genetic material from each parent is passed onto the child chromosome. The new chromosome produced is entered into the offspring pool. This new chromosome may represent an unexplored point in the search space. The mutation operator takes each chromosome in the offspring pool and randomly change part of its genetic make-up, i.e., its content. The probability of mutation occurring on any chromosome is determined by the user specified mutation rate. Chromosomes mutated or otherwise, are put back into the offspring pool after the mutation process. Thus each new generation of chromosomes are formed by the action of genetic operators (reproduction, cross-over and mutation) on the older population. Finally, the members of the population pool are compared with those of the offspring pool. The chromosomes are compared via their fitness value to derive a new population, where the weaker chromosomes may be eliminated. In exact, weaker members in the population pool are replaced by the better child chromosomes from the offspring pool. The heuristic for assessing the survival of each chromosome into the next generation is called the replacement strategy. The process of reproduction, cross-over, mutation and formation of a new population completes one generation cycle. A GA is left to progress through generations, until certain criteria (such as a fixed number of generations, or a time limit) are met. GAs were initially used for machine learning systems, but it was soon realised that GAs have great potential in function optimisation.

Figure 7 A block diagram for GA



4.2.3 Block diagram

A block diagram for GA describe in Figure 7.

4.2.3.1 Initialisation

The first step in the functioning of a GA is the generation of an initial population. Each member of this population encodes a possible solution to a problem. After creating the initial population, each individual is evaluated and assigned a fitness value according to the fitness function. Traditionally, the population is generated randomly, allowing the entire range of possible solutions (the *search space*). Occasionally, the solutions may be 'seeded' in areas where optimal solutions are likely to be found.

Seeding: Alternatively values can be used which the user believes are in the right area of the search space to find an optimum faster. Normally the seeding is performed by random selection which means that random data are generated. If a user has some knowledge of the search space, he can seed some values in the initial population that

are more likely to produce the global optimum solution quickly. This seedlings up the process of GAs.

4.2.3.2 Reproduction (selection)

Reproduction is usually the first operator applied on population. From the population, the chromosomes are selected to be parents to crossover and produce offspring. According to Darwin's evolution theory 'survival of the fittest' – the best ones should survive and create new offspring. This ensures that only the best characteristics are transmitted from the current generation to the next generation. The output of selection is mating pool that contains the chromosomes that mate with each other to generate off springs.

The selection occurs with a selection probability, *pselect*. The value of *pselect* is given as:

$$pselect(i) = fitness(i) / sum(fitness\ of\ all\ chromosomes\ in\ a\ population)$$

There are various methods for reproduction. Most commonly used are:

1 roulette wheel

2 tournament selection

3 Boltzmann selection

4 rank selection

5 steady state selection.

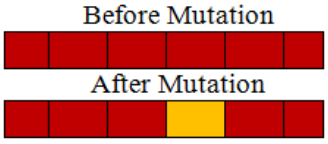
4.2.3.3 Crossover

During crossover, the two parents (chromosomes) exchange sub string information (genetic material) at a random position in the chromosome to produce two new strings (offspring). The crossover operators search for better genes (building blocks) within the genetic material. The objective here is to create better individuals and a better population over time by combining material from pairs of (fitter) members from the parent population (Domnguez-Jimnez et al., 2011; Spears and Anand, 1991). The result of this step is temporarily put into the offspring pool.

4.2.3.4 Mutation

Randomly change one or more digits in the string representing an individual. Figure 8 depicts single mutation by flipping one digit.

Figure 8 Mutation (see online version for colours)



These processes ultimately result in the next generation population of chromosomes that is different from the initial generation. Generally the average fitness will have increased by this procedure for the population, since only the best organisms from the first generation are selected for breeding, along with a small proportion of less fit solutions (Mishra et al., 2010), for reasons already mentioned above.

Although Crossover and Mutation are known as the main genetic operators, it is possible to use other operators such as regrouping, colonisation-extinction, or migration in GAs.

Non-lethal mutations accumulate within the offspring pool and increase the amount of genetic variation. The abundance of some genetic changes within the offspring pool can be reduced by natural selection, while other 'more favourable' mutations may accumulate and result in adaptive changes (Sharma and Bajpai, 2012).

4.2.3.5 Termination

This generational process is repeated until a termination condition has been reached. Common terminating conditions are:

- a solution is found that satisfies minimum criteria
- fixed number of generations reached
- allocated budget (computation time/money) reached
- the highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- manual inspection.

5 Literature study

There was a variety of GAs being introduced in the field of search, optimisation and machine learning. These were by far the best introduction to GAs and were becoming an important tool in machine learning and functional optimisation in Goldberg (1989). Method to optimise software testing efficiency has been evolved using GA. This method focused more on critical part clusters in the programmes, i.e., those parts that were the most critical and prone to errors. Exhaustive software testing was very rare, as it being becomes too costly in terms of money, time and efforts, even for a medium sized programme. So, only a part of programme was usually tested which may not be the most error prone. So, the approach to test by selecting the most error prone parts of programme may lead to increased testing efficiency by Xanthakis et al. (1992).

In 1997 it was often desirable to find test inputs that exercise special programme features. It was very difficult to do it by hand as it was very time consuming, especially when the software was complex and programme is large. Both random test data generation method and GA method was used for finding out the test inputs to the programme. The comparison between the two inputs obtained by the two techniques proved GA method to be the best solution for generating effective and good test data inputs by Michael et al. (1997).

In 1998 there were two problems widely observed in the study of optimisation framework, which were: testing specification failure and exception condition generation. So, a generalised test data generation framework was outlined based upon optimisation techniques. This framework could incorporate any number of testing criteria, for both functional and non-functional properties (Tracey et al., 1998).

Development history

EC	=	GP	+	ES	+	EP	+	GA
Evolutionary computing		Genetic programming		Evolution strategies		Evolutionary programming		Genetic algorithms
Rechenberg 1960		Koza 1992		Rechenberg 1965		Fogel 1962		Holland 1970

A tool and techniques were presented for test data generation and infeasible path identification. This tool was based on the Dynamic Technique and search using GAs, which introduced a new fitness function that combines control and data flow dynamic information to improve the process of search for test data. The unfeasibility issue was addressed by monitoring the GA's search progress. This approach was applicable indistinctively to non-linear paths too (Marcos et al., 2000; Miller et al., 2006; Pargas et al., 1999).

The use of GAs for automatic software test data generation was extended on the previous work on dynamic test data generation where the problem of test data generation was reduced to one of minimising a function (Michael et al., 2001; Pargas et al., 1999). GADGET, a new software test generation system that uses combinatorial optimisation to obtain condition/decision coverage of programmes was introduced. The GADGET system was fully automatic and supports all C/C++ language constructs. Certain issues which were discussed here, had not been discussed in earlier work on test-data generation, which concentrated on small programmes (most often single functions) written in restricted programming languages. The results thus obtained revealed that GA achieved maximum branch coverage among all other techniques (Michael et al., 1997; Michael and McGraw, 1998).

In 2003 Two GP-based procedures was introduced for selection and evaluation of test data. These procedures were related to two important questions: how to select test cases to reveal as many faults as possible and how to evaluate a test set and end the test. A tool called GPTest was used. The results obtained from this tool show the applicability of this approach by Claudia and Regina (2003).

A dynamic approach could handle array, pointer, function and other dynamic constructs more accurately than a static approach but it may also be more expensive since the programme under test was executed repeatedly. Automatic test generator (ATG), i.e., genet, produces test data for branch coverage with simpler instrumentation than branch functions, did not use programme graphs, and it was programming language independent. Genet used a GA to search for tests and formal concept analysis (FCA) to organise the relationships between tests and their execution traces. The relationships that genet learnt provides useful insights for test selection, test maintenance and debugging for automated test data generation technique in Khor and Grogono (2004).

A technique was proposed using a GA to automatically produce test cases for the unit testing of classes in a generic usage scenario. Test cases were described by chromosomes, which included information regarding which objects to create, which methods to invoke and which values to use as inputs. The proposed algorithm mutates them with the aim of maximising a given coverage measure by Tonella (2004).

Evolutionary testing (ET) automatically generated test data with good quality and was an effective technique based on evolutionary algorithm. A flag cost function was introduced which is the main component of fitness function in ET, whose value changes with the variation of flag problem. Based on that, a unified fitness calculation rule for flag conditions was proposed. In Liu et al. (2005), the experiments on programmes with flag problems and the traffic alert and collision avoidance system (TCAS) code showed the effectiveness of this unified approach.

A unique species per path ATDG technique was introduced which tackled path problems through goal-oriented search-based approach. The approach transforms the programme under test into a version in which multiple paths to search target were factored out. It showed as to how the paths to target can be assigned to species statically on the basis of slicing and transformation. The factoring out of paths results in several individual search landscapes, with feasible paths giving rise to landscapes that were potentially more conducive to test data discovery than the original overall landscape. The evolutionary searches were performed here with the publicly available genetic and evolutionary algorithm toolbox (GEATbox) by McMinn et al. (2006).

Automated test data generation played an important part in reducing the cost and increasing the reliability of software testing. However, a challenging problem in path-oriented test data generation was the existence of infeasible programme paths, where considerable effort may be wasted in trying to generate input data to traverse the paths. A heuristic-based approach was introduced for infeasible path detection for dynamic test data generation. This technique was based on the fact that many infeasible paths may have some properties in common. The technique detects the infeasible paths early in test data generation with high degree of accuracy, using the study done on those properties of infeasible paths in earlier studies in Ngo and Tan (2008).

A technique was proposed for generating test cases with a combination of path coverage testing criteria and GA. This technique usually tackles with infeasible path problem. This technique considered the edge of control flow graphs as assigned weights. The path with maximum value of the sum of all the assigned weights along a path, which is the value of fitness function for that path, was the most critical. Hence, this technique leads to execution of the most critical path by Srivastava and Kim (2009) and Satpathy et al. (2011).

Two versions of a technique based on scatter search meta heuristic technique for automated test data generation using branch coverage, were compared and analysed. These two versions were: TCSS and TCSS-LS. The first test case generator, called TCSS, used a diverse property to extend the search of test cases to all branches of the programme under test in order to generate test cases that cover these. The second, called TCSS-LS, was an extension of the previous test case generator which combines the diversity property with a local search method that allows the intensification of the search for test cases that cover the difficult branches. The results of comparison revealed that TCSS-LS perform better than any other test data generators (Rauf and Anwar, 2010).

In Rauf and Anwar (2010), another technique to generate software test data is presented using GUI-based test criteria. GUI applications were event driven. This technique uses GA to generate software test data. The sequence of events represented the candidate test case values. Event-flow graph was prepared for GUI applications, just like the control-flow graph (Kanmani and Maragathavalli, 2010a).

ET designated the use of meta heuristic search methods for test case generation. Search-based software engineering has been applied successfully to many software engineering activities ranging from requirement engineering to software maintenance and quality (assurance) assessment. Meta heuristic search methods were based on techniques such as evolutionary algorithms, simulated annealing, and tabu search which were generally used for test case generation. The comparison between GA testing and traditional random testing when performed, showed that GA-based testing was the best approach for testing by Kanmani and Maragathavalli.

In 2011 another technique was proposed by Malhotra and Garg (2011) based on adequacy-based test data generation, which uses the concept of mutation analysis to be applied at the time of test data generation only. This saved significant amount of time as the total time taken when mutation analysis was applied after the test data generation was the sum of time to generate test data and the time to apply mutation analysis to the generated test data. Then GAs were used to examine for the optimum solution. When adequacy-based technique was compared with path testing technique, it was found out that adequacy-based technique was better than path testing technique in terms of number of test cases generated and the time taken to generate those test cases (Lee et al., 2011; Chen and Zhong, 2008).

Lee et al. proposed one technique which was more effective fitness evaluation technique as compared to search-based software testing and meta heuristic search methods. This was called fitness evaluation programme (FEP). FEP, derived from a path constraint of SUT, was introduced as a special programme for evaluating fitness values to implement a test generation tool, named ConGA. It was used to generate test cases for C programmes for evaluating efficiency of the FEP-based test case generation technique. The experiments showed that the proposed technique could reduce significant amount of test data generation time on average (Bueno et al., 2011a).

In Bueno et al., a new test data generation technique was proposed which used the concept of diversity of test sets as a basis for the diversity-oriented test data generation (DOTG). Using DOTG, the testing quality was assumed to be greatly enhanced. A meta heuristic was described which can be used to automate the generation of test sets for DOTG testing technique: simulated annealing; a GA; and a proposed Meta heuristic named stimulated repulsion. The efficiency of performance of this technique was then evaluated using Monte Carlo simulation (Ali et al., 2011; Edvardsson, 1999).

6 Analysis of current trends

RQ. 1 Which heuristic search-based approaches have been used for automated test data generation? Most popular heuristic search-based approaches found in the literature are GA, simulated annealing, hill climbing, etc.

RQ. 2 Which type of automated test data has been generated? Test data can be in a form of Test cases for some systems so, they could be in a form of sequence of methods, sequence of their parameters. Test data can be a complete programme that can be executed to test the SUT.

RQ. 3 For which testing level heuristic search-based testing approach have been used more frequently? According to the basic concept of software engineering level of testing are unit testing, integration testing, system testing, and acceptance testing. Acceptance testing is used for the validation process. During literature we analysis that maximum research papers focuses on the unit, integration and system testing

RQ. 4 For which testing strategies has heuristic search-based approach have been used more frequently? As per our literature study we conclude that the test strategy is defined by the source which is used to create a testing model and also the coverage criteria which are used to derive the test date form the testing models. The testing models found in literature are specification-based model, state machines, control flow graph, data flow graph, etc.

RQ. 5 What are the most used alternatives to which heuristic search-based approach for automated test data generation are compared? Efficiency and cost effectiveness are the major factor for comparison and these are considered on two bases, i.e., global search-based approach and local search-based approach. In global search-based approach GA and simulated annealing are considered the most and in

local search-based approach only hill climbing is to be consider.

RQ. 6 Which automated test data generation testing field is used more frequently? Complete literature survey this section focused on software testing field. Figure 9 depicts software testing fields available while automation of test data generation yearly and Table 2 depicts the summary of testing field according to the research study. Software testing field is basically indicates that which testing field is opted by author of research paper while focusing on the problem in automated test data generation and software testing field is subdivided in three parts they are:

- white box testing
- black box testing
- grey box testing.

RQ. 7 Which automated test data generation testing approaches is used more frequently? This survey include software testing approaches used by researchers for defining their issues and automated test data generation. Figure 10 depicts year wise line graph of software testing approach and Table 3 depict about year wise focus of researchers on different software testing approach. In automated test data generation software testing approaches are sub divided in three parts they are:

- path-oriented approach
- random-oriented approach
- goal-oriented approach.

RQ. 8 Which automated test data generation implementation technique is used more frequently? Survey also include the third category, i.e., software implementation technique for automated test data generation using heuristic-based approach. Figure 11 depicts a year wise used implementation technique in their study and Table 4 depicts a summary of the same. Implementation technique further sub divided in three parts they are:

- static implementation techniques
- dynamic implementation techniques
- hybrid implementation techniques.

7 Conclusions and future work

In this paper we perform and exclusive survey of heuristic searched-based approach for automated test data generation. At the end we conclude that complete survey successfully highlighted important facts about various aspects of heuristic search-based automated test data generation techniques. Furthermore the research revealed that in the software testing there are number of research opportunities in the field of automated test data generation techniques. Our Efforts has been made to address the topic well, offer a meaningful presentation of a rich collection of the data. It has already been mentioned that the year 2011 (last year of the selected period) and 2012 (the current year) have shown rising trends in publications related to the heuristic-based automated test data generation techniques. Therefore this effort of presenting a survey could be remarked as a timely support for research community. Our work may help the reader to select the area and study the concepts of related area. After an extensive literature survey the future challenges were identified with the involvement of search-based approach:

1 Improvement of code coverage – software computing complexity of automated test data generation is very high. Finding an optimise solution is becoming a need to derived for a better code coverage. So that it reduces the cost of testing process.

2 Efficient predicate constraint modification – usually problem occurs due to feasible or in feasible paths. Taking a decision related to best of suitable path according to our problem we have to generate a constrains and check their suitability. And lot many researcher focuses on non-linear constrains that works only for unit testing. Finally non-linear constrains create high complexity there for it is a basic need to make our constrains scalable.

3 Loop bounds and infeasible path detection – looping a common problem in a software programmes. Detecting infeasible path of loop is a big challenge also to make this automatic is not yet done.

4 UML-based object-oriented software testing – the UML is a technique for prescribe specification, visualisation and documentation. UML provide flexibility to a software tester. The basic idea is to test software whose design is modelled using UML. The UML-based testing is useful for model-based system testing of distributed, component-based systems. UML sequence diagram, state chart diagrams, UML communication diagram, class diagram, and activity diagram are used for test data generation where we require constraint solving. The main challenge here is to collect information from one or combined UML diagrams and store in an efficient data structure. Test specifications and test data are collected from the data structure.

5 Agile testing – the agile testing are used for communication, simplicity, feedback and iterations. We always have to take care a close relation between developer and tester. The testers help each other in finding the quick solution. The agile approach use feedback at every step from client. To perform Agile testing agile approach to the development is mandatory. Basically in agile approach the entire software is divided in small modules and then identifies the priorities of the module depending on user requirement. The number of iteration should be reduced.

6 Test data generation for recursive programmes and procedures/functions – automation is required for test data generation and we also have to focuses on adequacy-based data generation using heuristic search-based approach with proper procedure and function.

7 GUI testing – GUI testing required to automated all the implementation techniques so that user can easily implement all the required approaches. Making a tool for proposed concept so that it is easily access by others users.

Figure 9 Testing fields in automated test data generation (see online version for colours)

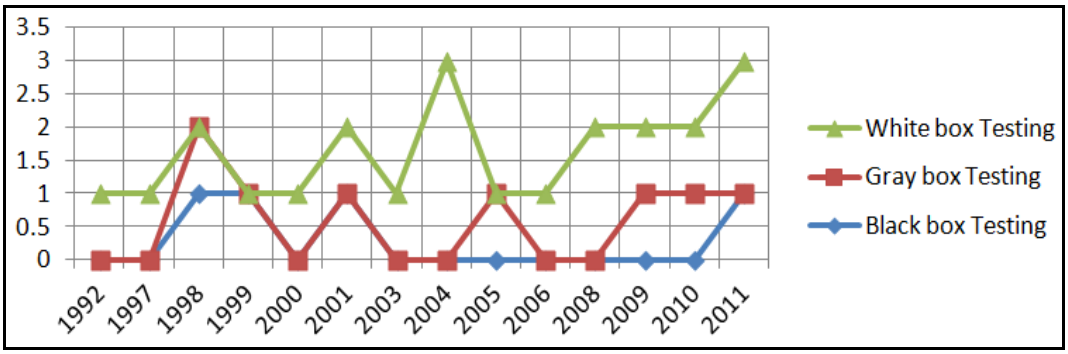


Table 2 Summary according to testing field

Sl. no.	*References*	*Author(s)*	*Pub. year*	*Testing field*
1	Xanthakis et al. (1992)	S. Xanthakis, C. Ellis, C. Skourlas, A. Le Gall, S. Katsikas and K. Karapoulios	1992	White box testing
2	Michael et al. (1997)	C. Michael, G. McGraw, M. Schatz and C. Walton	1997	White box testing
3	Tracey et al. (1998)	N. Tracey, J. Clark and K. Mander	1998	Grey box testing
4	Michael and McGraw (1998)	C. Michael and G. Mcgraw	1998	Black box testing
5	Jeng and Forgacs (1999)	B. Jeng and I. Forgacs	1999	Black box testing
6	Marcos et al. (2000)	P. Marcos, S. Bueno and M. Jino	2000	White box testing
7	Michael et al. (2001)	C. Christoph, C. Michael and G. McGraw	2001	Black box testing
8	Lin and Yeh (2001)	J. Lin and P. Yeh	2001	White box testing
9	Claudia and Regina (2003)	E. Claudia and V. Regina	2003	White box testing
10	Khor and Grogono (2004)	S. Khor and P. Grogono	2004	White box testing
11	Tonella (2004)	P. Tonella	2004	White box testing
12	Mansour and Salame (2004)	N. Mansour and M. Salame	2004	White box testing
13	Liu et al. (2005)	X. Liu, H. Liu, B. Wang, P. Chen and X. Cai	2005	Grey box testing
14	McMinn et al. (2006)	P. McMinn, M. Harman, D. Binkley and P. Tonella	2006	White box testing
15	Ngo and Tan (2008)	M. Ngo and H. Tan	2008	White box testing
16	Sofokleous and Andreou (2008)	A. Sofokleous and S. Andreou	2008	White box testing
17	Srivastava and Kim (2009)	P.R. Srivastava and T. Kim	2009	White box testing
18	Blanco et al. (2009)	R. Blanco, J. Tuya and B. Diaz	2009	Grey box testing
19	Rauf and Anwar (2010)	A. Rauf and S. Anwar	2010	Grey box testing
20	Kanmani and Maragathavalli (2010a)	S. Kanmani and P. Maragathavalli	2010	White box testing
21	Malhotra and Garg (2011)	R. Malhtora and M. Garg	2011	White box testing
22	Lee et al. (2011)	S. Lee, H. Choi, Y. Jeong, T. Kim, H. Chae and C. Chang	2011	Black box testing
23	Bueno et al. (2011a)	M. Bueno, M. Jino and W. Wong	2011	White box testing

Figure 10 Testing approaches in automated test data generation (see online version for colours)

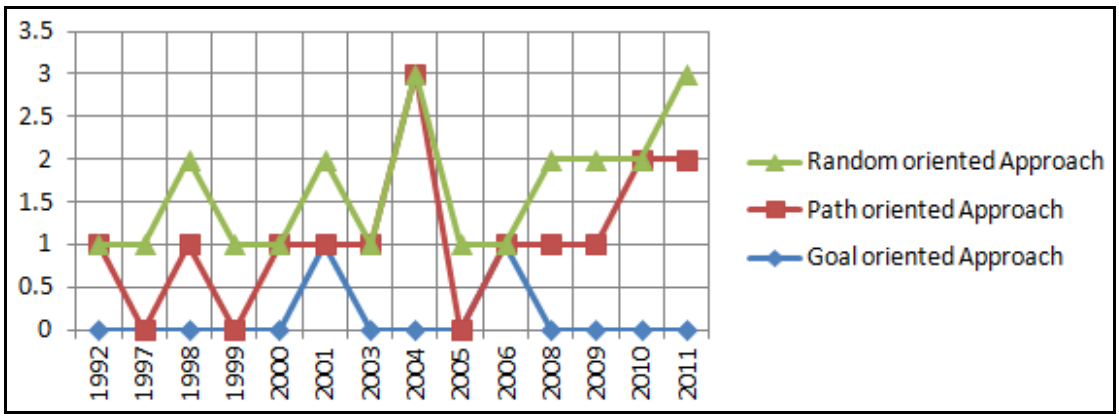


Table 3 Summary according to testing approaches

Sl. no.	*References*	*Author(s)*	*Pub. year*	*Testing approaches*
1	Xanthakis et al. (1992)	S. Xanthakis, C. Ellis, C. Skourlas, A. Le Gall, S. Katsikas and K. Karapoulios	1992	Path-oriented approach
2	Michael et al. (1997)	C. Michael, G. McGraw, M. Schatz and C. Walton	1997	Random-oriented approach
3	Tracey et al. (1998)	N. Tracey, J. Clark and K. Mander	1998	Path-oriented approach
4	Michael and McGraw (1998)	C. Michael and G. Mcgraw	1998	Random-oriented approach
5	Jeng and Forgacs (1999)	B. Jeng and I. Forgacs	1999	Random-oriented approach
6	Marcos et al. (2000)	P. Marcos, S. Bueno and M. Jino	2000	Path-oriented approach
7	Michael et al. (2001)	C. Christoph, C. Michael and G. McGraw	2001	Random-oriented approach
8	Lin and Yeh (2001)	J. Lin and P. Yeh	2001	Goal-oriented approach
9	Claudia and Regina (2003)	E. Claudia and V. Regina	2003	Path-oriented approach
10	Khor and Grogono (2004)	S. Khor and P. Grogono	2004	Path-oriented approach
11	Tonella (2004)	P. Tonella	2004	Path-oriented approach
12	Mansour and Salame (2004)	N. Mansour and M. Salame	2004	Path-oriented approach
13	Liu et al. (2005)	X. Liu, H. Liu, B. Wang, P. Chen and X. Cai	2005	Random-oriented approach
14	McMinn et al. (2006)	P. McMinn, M. Harman, D. Binkley and P. Tonella	2006	Goal-oriented approach
15	Ngo and Tan (2008)	M. Ngo and H. Tan	2008	Path-oriented approach
16	Sofokleous and Andreou (2008)	A. Sofokleous and S. Andreou	2008	Random-oriented approach
17	Srivastava and Kim (2009)	P.R. Srivastava and T. Kim	2009	Path-oriented approach
18	Blanco et al. (2009)	R. Blanco, J. Tuya and B. Diaz	2009	Random-oriented approach
19	Rauf and Anwar (2010)	A. Rauf and S. Anwar	2010	Path-oriented approach
20	Kanmani and Maragathavalli (2010a)	S. Kanmani and P. Maragathavalli	2010	Path-oriented approach
21	Malhotra and Garg (2011)	R. Malhtora and M. Garg	2011	Path-oriented approach
22	Lee et al. (2011)	S. Lee, H. Choi, Y. Jeong, T. Kim, H. Chae and C. Chang	2011	Path-oriented approach
23	Bueno et al. (2011a)	M. Bueno, M. Jino and W. Wong	2011	Random-oriented approach

Figure 11 Testing implementation technique in automated test data generation (see online version for colours)

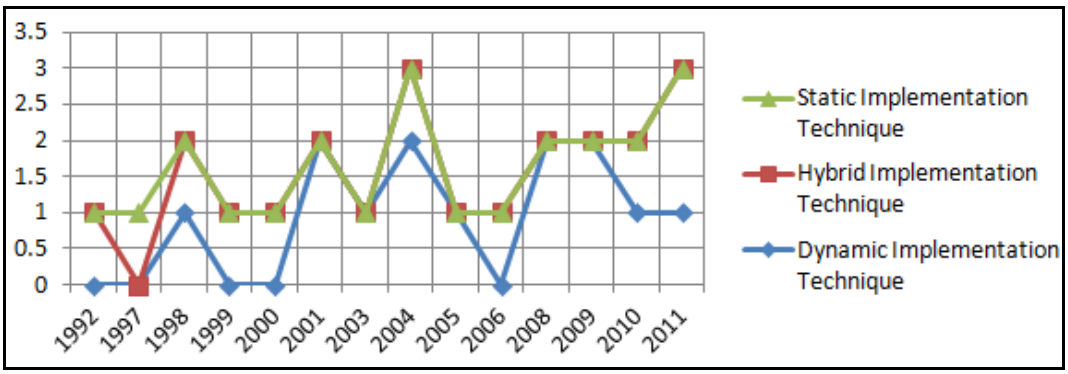


Table 4 Summary according to Implementation techniques

Sl. no.	*References*	*Author(s)*	*Pub. year*	*Implementation technique*
1	Xanthakis et al. (1992)	S. Xanthakis, C. Ellis, C. Skourlas, A. Le Gall, S. Katsikas and K. Karapoulios	1992	Hybrid implementation technique
2	Michael et al. (1997)	C. Michael, G. McGraw, M. Schatz and C. Walton	1997	Static implementation technique
3	Tracey et al. (1998)	N. Tracey, J. Clark and K. Mander	1998	Hybrid implementation technique
4	Michael and McGraw (1998)	C. Michael and G. Mcgraw	1998	Dynamic implementation technique
5	Jeng and Forgacs (1999)	B. Jeng and I. Forgacs	1999	Hybrid implementation technique
6	Marcos et al. (2000)	P. Marcos, S. Bueno and M. Jino	2000	Hybrid implementation technique
7	Michael et al. (2001)	C. Christoph, C. Michael and G. McGraw	2001	Dynamic implementation technique
8	Lin and Yeh (2001)	J. Lin and P. Yeh	2001	Dynamic implementation technique
9	Claudia and Regina (2003)	E. Claudia and V. Regina	2003	Dynamic implementation technique
10	Khor and Grogono (2004)	S. Khor and P. Grogono	2004	Dynamic implementation technique
11	Tonella (2004)	P. Tonella	2004	Hybrid implementation technique
12	Mansour and Salame (2004)	N. Mansour and M. Salame	2004	Dynamic implementation technique
13	Liu et al. (2005)	X. Liu, H. Liu, B. Wang, P. Chen and X. Cai	2005	Dynamic implementation technique
14	McMinn et al. (2006)	P. McMinn, M. Harman, D. Binkley and P. Tonella	2006	Hybrid implementation technique
15	Ngo and Tan (2008)	M. Ngo and H. Tan	2008	Dynamic implementation technique
16	Sofokleous and Andreou (2008)	A. Sofokleous and S. Andreou	2008	Dynamic implementation technique
17	Srivastava and Kim (2009)	P.R. Srivastava and T. Kim	2009	Dynamic implementation technique
18	Blanco et al. (2009)	R. Blanco, J. Tuya and B. Diaz	2009	Dynamic implementation technique
19	Rauf and Anwar (2010)	A. Rauf and S. Anwar	2010	Hybrid implementation technique
20	Kanmani and Maragathavalli (2010a)	S. Kanmani and P. Maragathavalli	2010	Dynamic implementation technique
21	Malhotra and Garg (2011)	R. Malhtora and M. Garg	2011	Hybrid implementation technique
22	Lee et al. (2011)	S. Lee, H. Choi, Y. Jeong, T. Kim, H. Chae and C. Chang	2011	Hybrid implementation technique
23	Bueno et al. (2011a)	M. Bueno, M. Jino and W. Wong	2011	Dynamic implementation technique

References

Ali, S. and Briand, L.C. (2010) 'A systematic review of the application and empirical investigation of search based test case generation', *IEEE Transactions on Software Engineering*, 0098-5589, Vol. 36, No. 6, pp.742–762.

Ali, S., Iqbal, M. Z., Arcuri, A. and Briand, L. (2011) 'A search based OCL constraint solver for model based test data generation', *11th International Conference On Quality Software*, IEEE, 978-1-4577-0754-4, pp.41–50.

Attol Unit Test [online] http//www.rational.com/products/testrt/index.jsp (accessed 18 April 2012), Rational Software.

Blanco, R., Tuya, J. and Diaz, B. (2009) 'Automated test data generation using a scatter search approach', *Information and Software Technology*, Vol. 51, No. 4, pp.708–720.

Bueno, M., Jino, M. and Wong, W. (2011a) 'Diversity oriented test data generation using metaheuristic search techniques', *Information Science*, Elsevier, 10.1016/j.ins.2011.01.025.

Bueno, M., Jino, M. and Wong, W. (2011b) 'Diversity oriented test data generation using metaheuristic search techniques', Elsevier Inc, 10.1016/j.ins.2011.01.025, 0020-0255.

Cantata [online] http://www.iplbath.com/p4.htm (accessed 6 February 2012), IPL.

Chen, Y. and Zhong, Y. (2008) 'Automatic path-oriented test data generation using a multi-population genetic algorithm', *Fourth International Conference on Natural Computation*, IEEE, 978-0-7695-3304-9, Vol. 1, pp.566–570.

Clarke, L.A. (1989) 'A system to generate test data and symbolically execute programs', IEEE *Transaction on Software Engineering*, 0098-5589, Vol. SE-2, No. 3, pp.215–222.

Claudia, E. and Regina, V. (2003) 'Selection and evaluation of test data based on genetic programming', *Software Quality Journal*, Vol. 11, No. 2, pp.167–186.

Domnguez-Jimnez, J.J., Estero-Botaro, A., Garca-Domnguez, A. and Medina-Bulo, I. (2011) 'Evolutionary mutation testing', *Information and Software Technology*, Elsevier, 10.1016/j.infsof.2011.03.008, Vol. 53, No. 10, pp.1108–1123.

Edvardsson, J. (1999) 'A survey on automatic test data generation', in *Proceedings of the Second Conference on Computer Science and Engineering in Linkoping*, pp.21–28.

Ferguson, R. and Korel, B. (1996) 'The chaining approach for software test date generation', *ACM Transactions on Software Engineering and Methodology*, 10.1145/226155.226158, Vol. 5, No. 1, pp.63–86.

Goldberg, D.E. (1989) *Genetic Algorithms: in Search, Optimization and Machine Learning*, Addison Wesley, MA.

Gong, D., Zhang, W. and Yao, X. (2011) 'Evolutionary generation of test data for many paths coverage base on grouping', *The Journal of System and Software Elsevier Inc*, 10.1016/j.jss.2011.06.028, Vol. 84, No. 12, pp.2222–2233.

Hannay, J.E., Sjberg, D.I.K. and Dyba, T. (2007) 'A systematic *review of theory use in software engineering experiments', IEEE* Transaction on Software Engineering, 0098-5589 Vol. 33, No. 2, pp.87–107.

Harman, M. and McMinn, P. (2007) 'A theoretical empirical analysis of evolutionary testing and hill climbing for structural test data generation', in *Proceedings of the 2007 International Symposium on Software Testing and Analysis (ISSTA '07)*, London, UK, ACM, 978-1-59593-734-6, pp.73–83.

Hennel, M.A., Woodward, M.R. and Hedley, D. (1989) 'Experience with path analysis and testing of programs', *IEEE Transaction on Software Engineering*, 0098-5589, Vol. SE-6, No. 3, pp.278–286.

Jeng, B. and Forgacs, I. (1999) 'An automatic approach of domain test data generation', *The Journal of Systems and Software*, Vol. 49, No. 1, pp.97–112.

Jones, B.F., Sthamer, H.H. and Eyres, D.E. (1996) 'Automatic structural testing using genetic algorithms', *Software Engineering Journal*, 0268-6961,Vol. 11, No. 5, pp.299–306.

Kanmani, S. and Maragathavalli, P. (2010a) 'Search-based software test data generation using evolutionary testing techniques', *International Journal of Software Engineering (IJSE)*, 0974-3162, Vol. 2, No. 1, pp.47–58.

Kanmani, S. and Maragathavalli, P. (2010b) 'Search-based software test data generation using evolutionary testing techniques', *International Journal of Software Engineering (IJSE)*, Vol. 1, No. 5, pp.10–22.

Khor, S. and Grogono, P. (2004) 'Using a genetic algorithm and formal concept analysis to generate branch coverage test data automatically', in *Proceedings of the 19th International Conference on Automated Software Engineering*, 0-7695-2131-2, pp.1068–3062.

Kitchenham, B. (2004) 'Procedures for performing systematic reviews', Keele University Technical Report and NICTA Technical Report 0400011T.1, TR/SE-0401.

Korel, B. and Al-Yami, A.M. (1996) 'Assertion-oriented automated test data generation', *IEEE Proceedings of ICSE-18*, 0-8186-7247-1, pp.71–80.

Korel, B., Ali, M. and Yami, A. (1996) 'Generating fast code from concurrent program dependent graphs', *Proceedings of ICSE-18*, IEEE, 1-58113-806-7,Vol. 39, No. 7, pp.175–181.

Lee, S., Choi, H., Jeong, Y., Kim, T., Chae, H. and Chang, C. (2011) 'An improved technique of fitness evaluation for evolutionary testing', *35th IEEE Annual Computer Software and Applications Conference Workshops*, 978-1-4577-0980-7, pp.190–193.

Lin, J. and Yeh, P. (2001) 'Automatic test data generation for path testing using gas', *Information Sciences*, Vol. 131, Nos. 1–4, pp.47–64.

Liu, X., Liu, H., Wang, B., Chen, P. and Cai, X. (2005) 'A unified fitness function calculation rule for flag conditions to improve evolutionary testing', in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, 1-58113-993-4, pp.337–341.

Ma, X., Li, J.J. and Weiss, D.M. (2007) 'Prioritized constraints with data sampling scores for automated test data generation', *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, 978-0-7695-2909-7, Vol. 11, No. 5, pp.1129–1134.

Mahmood, S. (2007) *A Systematic Review of Automated Test Data Generation Techniques*, Master Thesis Software Engineering.

Malhotra, R. and Garg, M. (2011) 'An adequacy based test data generation technique using genetic algorithms', *Journal of Information Processing Systems*, 10.3745/JIPS.2011.7.2.363, Vol. 7, No. 2, pp.363–384.

Mansour, N. and Salame, M., (2004) 'Data generation for path testing', *Software Quality Journal*, 10.1023/B:SQJO.0000024059.72478.4e, Vol. 12, No. 2, pp.121–136.

Marcos, P., Bueno, S. and Jino, M. (2000) 'Identification of potentially infeasible program paths by monitoring the search for test data', in *Proceedings of 15th International Conference on Automated Software Engineering*, 0-7695-0710-7, pp.209–218.

McCabe, T.J. (1989) 'A complexity measure', *IEEE Trans. on Software Engineering*, 0098-5589, Vol. SE-2, No. 4, pp.308–320.

McMinn, P. (2004) 'Search-based software test data generation: a survey', *Software Testing, Verification and Reliability*, 10.1002/stvr.294, Vol. 14, No. 2, pp.105–156.

McMinn, P., Harman, M., Binkley, D. and Tonella, P. (2006) 'The species per path approach to search based test data generation', in *Proceedings of the 6th International Symposium on Software Testing and Analysis*, 1-59593-263-1, pp.13–24.

Michael, C. and McGraw, G. (1998) 'automated software test data generation for complex programs', in *Proceedings of 13th International Conference on Automated Software Engineering*, 0-8186-8750-9, pp.136–146.

Michael, C., McGraw, G. and Schatz, A. (2001) 'Generating software test data by evolution', *IEEE Transactions on Software Engineering*, 0098-5589, Vol. 27, No. 12, pp.1085–1110.

Michael, C., McGraw, G., Schatz, M. and Walton, C. (1997) 'Genetic algorithms for dynamic test data generation', IEEE, 0-8186-7961, No. 7, pp.307–308.

Miller, J., Reformat, M. and Zhang, H. (2006) 'Automatic test data generation using genetic algorithm and program dependence graphs', *Information and Software Technology*, Vol. 48, No. 7, pp.586–605.

Mishra, K., Tiwari, S., Kumar, A. and Misra, A. (2010) 'An approach for mutation testing using elitist genetic algorithm', *2010 3rd International Conference on Computer Science and Information Technology*, pp.426–429.

Ngo, M. and Tan, H. (2008) 'Heuristics-based Infeasible path detection for dynamic test data generation', *Information and Software Technology*, Elsevier, Vol. 50, Nos. 7–8, pp.641–655.

Offutt, A., Pan, J., Tewary, K. and Zhang, T. (1995) 'An experimental evaluation of data flow and mutation testing', National Science Foundation under Grant CCR-93-11967.

Pargas, R.P., Harrold, M.J. and Peck, R.R. (1999) 'Test-data generation using genetic algorithms', *The Journal of Software Testing, Verification and Reliability*, Vol. 9, No. 4, pp.263–282.

Pohlheim, H. (2007) 'GEATbx – The genetic and evolutionary algorithm toolbox for Matlab', Matlab [online] http//www.geatbx.com/docu/index.html (accessed 3 March 2012).

Prather, R.E. and Myers, J.P. (1989) 'The path prefix software engineering', *IEEE Trans. on Software Engineering*, 0098-5589, Vol. SE-13, No. 7, pp.761–766.

Rauf, A. and Anwar, S. (2010) 'Automated GUI test coverage analysis using GA', *Seventh International Conference on Information Technology*, 978-1-4244-6270-4, pp.1057–1062.

Satpathy, M., Yeolekar, A., Peranandam, P. and Ramesh, S. (2011) 'Efficient coverage of parallel and hierarchical state flow models for test case generation', *Software Testing, Verification and Reliability*, Wiley Online Library.

Sharma, M. and Bajpai, N. (2012) 'Automatic generation and execution of mutants', *International Journal of Computer Applications*, (0975 – 8887), Vol. 44, No. 3, pp.6–12.

Sjberg, D.I.K., Hannay, J.E., Hansen, O., Kampenes, V.B., Karahasanovic, A., Liborg, N.K. and Rekdal, A.C. (2005) 'A survey of controlled experiments in software engineering', *IEEE Transactions on Software Engineering*, 0098-5589, Vol. 31, No. 9, pp.733–753.

Sofokleous, A. and Andreou, S. (2008) 'Automatic evolutionary test data generation for dynamic software testing', *The Journal of Systems and Software*, Elsevier, 10.1016/j.jss.2007.12.809, Vol. 81, No. 11, pp.1883–1898.

Spears, W.M. and Anand, V. (1991) 'A study of crossover operators in genetic programming', *Proc. 6th Int. Symp. on Methodologies for Intelligent Systems*.

Srivastava, P.R. and Kim, T. (2009) 'Application of genetic algorithm in software testing', *International Journal of Software Engineering and its Applications*, Vol. 3, No. 4, pp.87–96.

Tahbildar, H. and Kalita, B. (2010) 'Heuristic approach of automated test data generation for programs having array of different dimensions and loops with variable number of iteration', *International Journal of Software Engineering and Applications*, DOI: 10.5121/ijsea.2010.1405, Vol. 1, No. 4, pp.75–93.

TCAT [online] http://www.soft.com/TestWorks (accessed 27 March 2012), Software Research.

TESSY [online] http://www.ats-software.de/html/prod_tessy.htm (accessed 3 March 2012), Razorcat Development.

Tonella, P. (2004) 'Evolutionary testing of classes', *ISSTA '04 Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis*, 1-58113-820-2, pp.119–128.

Tracey, N., Clark, J. and Mander, K. (1998) 'Automated program flaw finding using simulated annealing', in *Proceedings of the 1998 ACM SIGSOFT International Symposium on Software Testing and Analysis '98*, 0-89791-971-8, Vol. 23, No. 02, pp.73–81.

Xanthakis, S., Ellis, C., Skourlas, C., Le Gall, A., Katsikas, S. and Karapoulios, K. (1992) 'Application of genetic algorithm to software testing', in *Proceedings of 5th International Conference on Software Engineering and its Applications*, Toulouse, France, pp.625–636.