CHAPTER-4

4.1 HISTORICAL BACKGROUND

For the quality of the test, the designing of test cases are important. A large number of test methods have been developed to support the developer when choosing appropriate test data. Some useful testing methods are structural testing methods, functional testing methods and statistical testing methods [81]. It is very difficult to develop correct, good and unique test cases manually. Therefore, automation of test cases is important. The success of a test data generation method largely depends upon the efficiency of its search technique. Different researchers have worked on test case automation from time to time with the aim to increase the quality of the tests and to achieve substantial cost saving in the system development by means of higher degree of automation.

One critical task in software testing is the creation of test data to satisfy a given test-coverage criterion. This process is called as Test Data Generation. Developments in the field of ‘automated test data generation’ were initiated in early 70’s when papers on “Testing large software with automated software evaluation systems” by Ramamoorthy, in 1976 [6] and Holland, in 1975 [8] and “Automatic Generation of Floating-Point Test Data” by Miller and Spooner, in 1976 [10] is published. Work done by Nevertheless, Clarke in 1976 [5] is considered to be the first of its kind to produce a solid algorithm for Automatic Test Data Generation (ATDG).

Various mechanisms exist to contextualize complex testing problems with respect to existing literature. Problem classification is an important prerequisite to the selection of a suitable solution strategy since information regarding problem complexity and existing algorithms provide useful points of departure for new algorithm development. Automatically test data generation for software testing with minimum time and cost is a known to be NP-hard and only exhaustive search guarantees the optimal solutions. But these can become prohibitively expensive to compute even for small problems. Several methods have been used to solve combinatorial optimization problem but each of them has its own limitation and
advantages. Some useful existing optimization techniques to solve the software testing problems have surveyed. Some of the important literature on software test case generation for software testing has been presented in respect of techniques ranging from the traditional exact methods to modern metaheuristic methods here.

4.2 SOFTWARE COVERAGE ANALYSIS TECHNIQUES

A number of test-data generation techniques have been developed for coverage of software under test. Each one of them uses different kinds or variations of existing testing techniques [23, 34, 83, 84]. Test adequacy criterion usually involves coverage analysis, which can be measured based on different aspects of software like statements, branches, paths and all-uses [53].

In statement testing, each and every statement of the software/program under test has to be executed at least once during testing. The main drawback of statement testing is that even if one achieves a very high level of statement coverage, it does not reflect that program is error free [34].

Branch coverage is stronger testing criteria than statement coverage testing criteria [39]. For branch coverage each and every branch has to be executed at least once during testing. In this testing all control transfer are executed [39]. However some errors can only be detected if the statements and branches are executed in a certain order [34].

Path testing searches the program domain for suitable test cases that covers every possible path in the Software Under Test (SUT) [53]. It is stronger criteria as compare to statement and branch coverage criteria [37]. This tries to find out the percentage of code coverage to more extent and hence increase the chances of error detection [34]. However, it is generally impossible to achieve this goal, for several reasons. First, a program may contain an infinite number of paths when the program has loops [23, 34, 53, 85]. Second, the number of paths in a program is exponential to the number of branches in it [83, 86] and many of them may be unfeasible. Third, the numbers of test cases are too large, since each path can be covered by several test cases. For these reasons, the problem of path testing can become a NP complete problem [83], making the covering of all possible path computationally impractical. Since it is impossible to cover all paths in software, the problem of path testing selects a subset of paths to execute and find test data to cover it.
Frankl, in 1988 [113] uses all-uses criteria in her paper “An Applicable Family of Data Flow Testing Criteria”. This was stronger criteria as compare to already discuss one. It focuses on all p-uses and c-uses of each and every variable hence coving each and every path and branch of software under the test.

Girgis [13] has proposed a technique that uses GA which is guided by the data flow dependencies in the program to search for test data to fulfill data flow path selection criteria namely the all-uses criterion. Data-flow testing is important because it augments control-flow testing criteria and concentrates on how a variable is defined and used, which could lead to more efficient and targeted test suites. Girgis used the ratio between the numbers of the covered def-use paths covered by a test case to the total number of def-use paths. This technique cannot find the closeness of the test cases because the fitness function gives the same value for all test cases that cover the same number of def-use paths and ‘0’ for all test cases that do not cover any def-use path. This technique will result in a loss of valuable information (test data that contains good genes) when it ignores test cases that cover only the use node [25].

**4.3 TESTING TYPES AND APPROACHES**

Various test data generation methods have been proposed in the literature. These methods can be classified as Static methods, Dynamic methods, functional methods, random test data generators, symbolic evaluators and function minimization methods [87].

**4.3.1 Static Testing**

The static methods never require the execution of code on computers but involve the tester to go through the code to find the errors. The first automatic test generation approach proposed by Clarke in 1976 was static which based on symbolic execution [5]. The symbolic execution methods are static, in the sense that they analyze a program to obtain a set of symbolic representations of each condition predicate along a selected path. The expressions are obtained by attributing symbolic values to the input variables. If the predicates are linear, then the solution can be obtained by using linear programming [88]. Symbolic Test Data Generation Techniques assign symbolic values to variables to create algebraic expressions for the constraints in the program and use a constraints solver to find a solution for these expressions that satisfies a test requirement [7, 11]. Symbolic execution cannot
determine which symbolic value of the potential values will be used for array or pointer. Symbolic execution cannot find floating point inputs because the current constraint solvers cannot solve floating point constraints.

Constraint Based Testing builds up constraint systems which describe the given test goal. The solution to this constraint system brings about satisfaction of the goal. The original purpose of Constraint Based testing was to generate test data for mutation testing. Reachability constraints within the constraint system describe conditions under which a particular statement will be reached. Necessity constraints describe the conditions under which a mutant will be killed. With Constraint-based testing, constraints must be computed before they are analysed.

Another test data generation technique type of Constraint-Based Testing is invented by DeMillo and Offutt [2] and based on symbolic execution is used to develop the constraints in terms of the input variables is called Domain Reduction. Domain Reduction is then used to attempt a solution to the constraints. The first step of this technique starts with the finding of domains of each input variable which are derived from type or specification information or be supplied by the tester. The domains are then reduced using information in the constraints, beginning with those involving a relation operator, a variable, a constant and constraints involving a relation operator and two variables. This helps in reducing the search space (input domain) for solving a constraint system. Remaining constraints are then simplified by back-substituting values. Although efforts were made for improving the performance of algorithmic search methods by employing some techniques likes identification of undesirable variables, finding optimum order of consideration of input variables, use of binary search algorithm and expression handling technique but these required a plenty of manual and time consuming analysis. This makes algorithmic search methods very slow and ineffective. These algorithms also lack global search capabilities which are a necessary requirement for software testing where objective-functions are very complex and usually non-linear [82]. Since these constraints are derived using symbolic execution, the method suffers from similar problems involving loops, procedure calls and computed storage locations [88].

To overcome the limitations of domain reduction method another method called Dynamic Domain Reduction was introduced by Offutt, in 1997 [11]. Dynamic Domain Reduction also starts with the domain of input variables like the Domain
Reduction but these domains are reduced dynamically during the Symbolic Execution stage, using constraints composed from branch predicates encountered as the path is followed. If the branch predicate involves a variable comparison, the domains of the input variables responsible for the outcome at the decision are split at some arbitrary split point rather than assigning random input values. Dynamic Domain Reduction still suffers with difficulties due to computed storage locations and loops. Furthermore, it is not clear how domain reduction techniques handle non-ordinal variable types such as enumerations [88].

4.3.2 Dynamic Testing

Unlike the static testing, dynamic methods require the execution of code. The test cases are run on the code of the software product that has to be tested with the help of computer. Since array subscripts and pointer values are known at run-time, many of the problems associated with symbolic execution can be discovered with dynamic methods which are not possible with static testing. Dynamic Test Data Generation Technique collects information during the execution of the program to determine which test cases come closest to satisfying the requirement. Then, test inputs are incrementally modified until one of them satisfies the requirements [3, 9].

Random Test-Data Generation Techniques select inputs at random until useful inputs are found [14]. In random testing, random values are generated from domains of inputs and program is executed using these values. If these inputs are able to satisfy the testing criterion then they form a test case [82]. This technique may fail to find test data to satisfy the requirements because information about the test requirements is not incorporated into the generation process [25]. J. W. Duran and S. Ntafos in 1984 [64] reported random testing to be satisfactory for small as well as large programs. Thayer and others [89] used it to measure reliability of the system. Demillo and others [90] also used random testing for identifying seeded faults in programs.

Mayer and Schneckenburger [91] empirically investigated different flavors of adaptive random testing. They concluded that distance based random testing and restricted random testing are the best methods for this class of testing techniques. This approach is quick and simple but it is a poor choice with complex programs and complex adequacy criteria. The probability of selecting an adequate input by chance could be low in this case. The biggest issue for random approach is that of adequate
test data selection. Myers [37] viewed random testing as a worst case of program testing.

The results of actual executions of the program with a search technique were first studied by Miller and Spooner [30]. These were originally designed for the generation of floating-point test data. However, the principles are more widely applicable. The tester selects a path through the program and then produces a straight-line version of it, containing only that path. Korel suggested a dynamic approach to automatic test data generation using function minimization and directed search [32]. In this work, the test data generation procedure worked on an instrumented version of the original program without the need for a straight-line version to be produced. The search targeted the satisfaction of each branch predicate along the path in turn, circumventing issues encountered by the work of Miller and Spooner. In this type exploratory search is done, in which the selected input variables are modified by a small amount and submitted to the program. Korel [32] used alternate variable method for its dynamic test data generator. The alternate variable method works in two phases. First, an input variable is selected and its value is changed in small steps just to find out the direction in which variable minimizes the branch function. This is called exploratory search. Once the direction of search is known then pattern search is taken in large steps to find the value of the variable in consideration for satisfying or minimizing the branch function. If selected value of the variable fails to decrease the branch function then steps of the pattern search are decreased successively before exploring other variables for minimization purpose. Gallagher and Narasimhan [95] built on Korel's work for programs written in ADA. In particular, this was the first work to record support for the use of logical connectives within branch predicates. Dynamic techniques can stall when they encounter local minima because they depend on local search techniques such as gradient descent [82].

Korel, in 1992 [95] was first used concept of Goal-Oriented Approach. In 1992, Goal-oriented techniques identify test data covering a selected goal such as a statement or a branch, irrespective of the path taken [23]. This approach involves two basic steps: to identify a set of statements (respective branches) the covering of which implies covering the criterion; to generate input test data that execute every selected statement (respective branch) [96]. Two typical approaches, Assertion-Based and Chaining Approach are known as goal oriented. In the first case assertions are inserted
and then solved. In chaining approach data dependence analysis is carried out. It uses the concept of an event sequence as an intermediate means of deciding the type of path required for execution up to the target node [97, 99]. An event sequence is basically a succession of program nodes that are to be executed. The initial event sequence consists of just the start node and target node. Extra nodes are then inserted into this event sequence when the test data search encounters difficulties. Generally the goal-oriented approach faces issues of goal selection and selection of adequate test data [98].

4.3.3 Functional Testing

Functional Testing is also called as specification based or Black Box Testing. If testers want to test functional requirements, they may use Black-Box Testing technique. On the other hand, function minimization methods are dynamic. They are based on program execution. Black Box Testing does not need knowledge of how software is programmed. It generates test data for software from its specification without considering the behavior of the program under test. Testers inject test data to execute program, then compare actual result with the specified test oracle. The test engineers engaged in black box testing only knows the sets of input and expected output and is unaware of how those inputs are transformed into output by software. Black box testing requires functional knowledge of the product to be tested [1, 9]. Black Box Testing helps in the overall functionality verification of the system under test.

Syntax Based Testing involves on boundary value analysis, partition analysis, domain testing, equivalence partitioning, domain partitioning, and functional analysis [100, 101, 102, 87]. Hoffman in 1999 [100] presented a technique based on boundary values analysis in this technique the relationship between a generalized approach to boundary values and statement coverage is explored. Jeng in 1999 [101] has presented a technique that is mainly related to domain testing. It combined the static approach with the dynamic search method. In 1997, Gallagher and Lakshmi Narasimhan [102] proposed a method for locating input domain boundaries intersections and generating ON/OFF test data points.
4.4 METAHEURISTICS

Metaheuristics are general heuristic methods that guide the search through the solution space, using as surrogate algorithms some form of heuristics and usually local search. Starting from an initial solution built by some heuristic, metaheuristics improve it iteratively until a stopping criterion is met. The stopping criterion can be elapsed time, number of iterations, number of evaluations of the objective function and so on [41]. Voss in 1999 [105] described a metaheuristic as “Iterative master processes that guides and modifies the operations of subordinate heuristics to efficiently produce high quality solutions”.

The most successful search algorithm class is based on metaheuristic techniques like Hill Climbing (HC), Tabu Search (TS), Simulated Annealing (SA), Genetic Algorithm (GA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Cat Intelligence etc. McMinn [88] has provided a detail and up to date survey on use of metaheuristic techniques for software testing. Several metaheuristics have been suggested for path coverage [83, 85], statement coverage [23] and branch coverage [23, 34]. In such cases the use of metaheuristics would be very useful in providing usable results in a reasonable time.

4.4.1 Hill Climbing (HC)

Hill Climbing is a local search algorithm. Starting from a solution created at random or by some problem specific heuristic, standard local search tries to improve on it by iteratively deriving a similar solution in the neighborhood of the so-far best solution. Responsible for finding a neighboring solution is a move-operator, which must be carefully defined according to the problem. This progression improvement is likened to the climbing of hills in the landscape of a maximising objective function [88]. It applies standard local search multiple times from different starting solutions and returns the best local optimum identified [41]. The major disadvantage of standard local search is its high probability of getting trapped at a poor local optimum.

4.4.2 Tabu Search (TS)

Diaz [104] developed a tabu search based test generator that have used program control flow graph for branch coverage. It maintains a search list also called as tabu list. This strategy extends local search by the introduction of memory. Stagnation at a local optimum is avoided by maintaining a data structure called
history, in which the last created solutions or alternatively the last moves (i.e., changes from one candidate solution to the next) are stored. These solutions, respectively moves, are forbidden (tabu) in the next iteration and the algorithm is forced to approach unexplored areas of the search space. It uses neighborhood information and backtracking for solving local optima. They defined two cost functions for intensifying and diversifying the search mechanism. These cost functions are similar to the functions used by Wegner, in 12002 [81] in which individuals are penalised for taking wrong path while executing the program. Penalty is fixed on the basis of error value produced by an individual in the effort of satisfying a branch constraint.

4.4.3 Simulated Annealing (SA)

Another way for enabling local search to escape from local optima and approach new areas of attraction in the search space is to sometimes also accept worse neighboring solutions. Simulated annealing does this in a probabilistic way. Simulated Annealing (SA) algorithms, based on the analogy of annealing process of metals, were proposed by Metropolis [106] in 1953 and were first applied to combinatorial optimization problems by Kirkpatrick in 1983 [107]. SA is considered to be an improvement heuristic where a given initial solution is iteratively improved upon. SA is a metaheuristic method used for test case generation in which process of cooling of a material simulates the change in energy level with time or iterations. The steady state in energy symbolizes the convergence of solution. At the beginning of the optimization, worse solutions are accepted with a relatively high probability and this probability is reduced over time in order to achieve convergence. A number of researchers have applied SA to testing problems. Tracey [108, 109] constructed a SA based test data generator for safety critical system. A hybrid objective function is used which includes concepts, branch distance and number of executed control dependent nodes. N. Mansour [83] in 2004 presents that GA is faster than SA for generating test cases.

4.4.4 Genetic Algorithms (GAs)

GA is one of the most popular and intensively pursued techniques for software testing. The GA is a global search metaheuristic proposed originally by Holland [8] in 1975. Extensive work has been done on the development of the original algorithm in
the last 20 years and it has been applied successfully in many fields of science and engineering [92, 93]. The GA is based on the principles of Darwin’s theory of natural evolution and belongs to a more general category, the Evolutionary Algorithms (EAs).

Recently, test-data generation techniques based on genetic algorithms (GAs) have been developed [13, 24, 23, 81, 84, 103]. Whereas previous techniques may not be useful in practice, techniques based on GAs have the potential to use for real systems. Xanthakis [103] first time applied GA for automatic test case generation. Pargas et al. [23] presented a Genetic Algorithm directed by the control-dependence graph of the program under test to search for test data to satisfy all-nodes and all-branches criteria. Wagener [18] logarithmized the objective function to provide better guidance for its GA based test case generator. They present a test environment for automatic generation of test data for statement and branch testing. These techniques evolve a set of test data using genetic operations (selection and recombination) to find the required test data. Michael et al. [24] used GAs for automatic test-data generation to satisfy condition-decision test-coverage criterion. They proposed a GA based test generation system called Genetic Algorithm Data Generation Tool (GADGET) to generate test cases for large C and C++ programs by using condition decision coverage metrics.

Watkins [84] and Ropar [38] used coverage based criteria for assessing the fitness of individuals in their GA based test generator. Lin and Yeh [85] used hamming distance based metric in objective function of their GA program to identify the similarity and distance between actual path and already selected target path in dynamic testing. Bouchachia [116] incorporated immune operators in genetic algorithm to generate software test data for condition coverage.

GA has started getting competition from other heuristic search techniques like Particle Swarm Optimization. Various works [16 - 20] show that particle swarm optimization is equally well suited or even better than Genetic Algorithms for solving a number of test problems [21].

4.4.5 Particle Swarm Optimization (PSO)

PSO has been applied successfully to a wide variety of search and optimization problems [16 -20, 110, 111]. It is motivated from the simulation of social
behavior [114]. PSO was proposed by Kennedy and Eberhart in 1995 [16] is commonly used to solve the problem of nonlinear optimization through the coordination between the individual to implement population convergence. Windisch [17] have reported the application of this swarm intelligence based technique for test data generation for dynamic testing. They have conducted experiments to prove the usefulness and utility of search algorithm towards test case generation. Compared with GA, PSO has some attractive characteristics. It has memory, so knowledge of good solutions is retained by all particles; whereas in GA, previous knowledge of the problem is destroyed once the population changes. It has constructive cooperation between particles, particles in the swarm share information between them. The individuals in the PSO update themselves using the best value of their own and the best value of the whole population in the history. Finally, the entire population will converge to the global optimum [115]. The research work of different researchers from time to time [16-20] shows that PSO is better alternates compare to GAs in generation of test cases.

4.4.6 Ant Colony Optimization (ACO)

ACO has been applied in the area of software testing in 2003 [80, 117]. Boerner and Gutjahr [80] described an approach involving ACO and a Markov software usage model for deriving a set of test paths for a software system. McMinn and Holcombe [117] presented ACO as a supplementary optimization stage for finding sequences of transitional statements in generating test data for evolutionary testing. H. Li and C. P. Lam [79, 118] proposed an ACO approach to test data generation for the state-based software testing. Ayari et al. [119] proposed an approach based on Ant Colony to reduce the cost of test data generation in the context of mutation testing. Srivastava and Rai [120] proposed an ant colony optimization based approach to test sequence generation for control-flow based software testing. K. Li et al. [121] presents a model of generating test data based on an improved ant colony optimization and path coverage criteria. P. R. Srivastava et al. [122] made an algorithm with the help of an ACO for the optimal path identification by using the basic property and behavior of the ants. This ACO based approach is enhanced by a probability density estimation technique in order to better guide the search for continuous input parameters.
4.4.4 Hybrid Metaheuristics

Hybridization of evolutionary algorithms with local search has been investigated in many studies [124 – 126]. Such a hybrid is often referred to as a memetic algorithm [127–129]. Talbi [123] gave a classification framework and taxonomy of hybrid metaheuristics.

L. Wang and D. Z. Zheng [135] present a hybrid approach which combined Genetic Algorithm and local optimization technique for simulation optimization problems. Through the combination of genetic algorithms with the local optimization method, it can maximally use the good global property of random searching and the convergence rate of a local method. Their study considers the sampling procedure based on orthogonal design and quantization technology, the use of orthogonal Genetic Algorithm with quantization for the global exploration and the application of local optimization technique for local exploitation. The final experimental results demonstrated that the proposed approach can find optimal or close-to-optimal solutions and is superior to other recent algorithms in simulation optimization.

D. Kusum, D. K. Nath [133] presented a Hybrid Binary Coded Genetic Algorithm (HBGA) for constrained optimization. They called it HBGA-C. It is based on the features of Hybrid Binary Coded Genetic Algorithms. The aim was to implement constraint handling technique to HBGA. It was easy to implement and it also provided feasible and better solutions with a fewer number of function evaluations. It was compared with Constrained Binary GA (BGA-C) by incorporating the constraint handling technique on BGA that used Roulette wheel selection and single point crossover. Their comparative performance was tested on a set of twenty five constrained benchmark problems. The results have shown the better performance.

Y. R. Ali, O. Nursel, K. Necmettin and O. Ferruh [131] describe a new hybrid approach, which deals with the improvement of shape optimization process. The objective is to contribute to the development of more efficient shape optimization approaches in an integrated optimal topology and shape optimization area with the help of GA and robustness issues. An improved GA is introduced to solve multi-objective shape design optimization problems. The specific issue is to overcome the limitations caused by larger population of solutions in the pure multi-objective genetic
algorithm. The combination of genetic algorithm with robust parameter design through a smaller population of individuals results in a solution that leads to better parameter values for design optimization problems. The effectiveness of the proposed hybrid approach is illustrated and evaluated with test problems. It shows that the proposed approach can be used as first stage in other multi-objective GA to enhance the performance of GA. Finally, the shape optimization is applied for solving multi-objective shape design optimization problems.

The social foraging behavior of bacteria has been used to solve optimization problems [130]. V. K. D. Hwa, A. Ajith and C. J. Hoon proposed a hybrid approach involving GA and Bacterial Foraging (BF) algorithms for function optimization problems. The algorithm emphasises on mutation, crossover, variation of step sizes, and the lifetime of the bacteria. The algorithm is then used to tune a PID controller of an Automatic Voltage Regulator (AVR). Simulation results show the efficiency of it. It could easily be extended for other global optimization problems.

Devraj [144] presented a GA with adaptive mutation based upon non-revisiting. The algorithm removed the duplicate individuals. Moreover, instead of using simple GA, by which the individuals are generated again and again, which is clearly wastage of time and computational resources, an improved GA has been suggested. The proposed GA is flexible with all function with any number of variables.

A hybrid algorithm based on Simulating Annealing and Genetic Algorithm was proposed by Wangsnd [135 -136] to improve neighbor search ability of the heuristic algorithms. They divided the initial population which was generated randomly into subpopulations and apply multiple crossover operations to these subpopulations in order to improve the exploring potential of traditional GA based approaches. They analyzed that this hybrid algorithm provides better results as compare to existing simple GA but hybrid heuristic is computationally more expensive. Using a hybrid of Ant System and Genetic Algorithm Noorul Haq [137, 138] proposed new techniques that give as compared to pure metaheuristics techniques. In this hybridization the output of Ant System became input GA. A hybrid algorithm based on Simulating Annealing, Genetic algorithm and iterative hill-climbing procedure to avoid local-minima at each step in the iteration is proposed in 2004 by Nearchou [139].
A more superior hybrid genetic algorithm in which initial solutions have been searched by PSO for multi-objective scheduling of flexible manufacturing system was proposed by Biswal [140]. The outstanding performance of this algorithm overcomes the main limitation of early work done by Naderi in 2009 [143].


K. Li in 2010 [142] proposed a GPSMA (Genetic-Particle Swarm Mixed Algorithm) to breed software test data for path testing. On the basis of population division, drawing on the idea of niche, the GPSMA method used to generate test data in each subpopulation. They used a new method to breed software test data called GPSMA for structure data test generation. They introduced a new strategy to replace the mutation operation in traditional GA. They used the “excellent rate of production” to implement the interaction between sub-populations. Theoretical analysis and practical testing results show that the approach is simpler, easier and more effective in generating test data automatically. The comparison with ant colony optimization and traditional genetic algorithm shows that the GPSMA is a good alternative for test data generation problems.

4.5 LIMITATIONS / GAPS OF EXISTING RESEARCH

After a comprehensive study made on the existing literature, a lot of limitations/gaps have been found in the area of Software Testing:

- Majority of work reported for software testing problems has been dealt with statement testing, branch testing, path testing, and data flow testing which have their own limitations. Hence a more attention is required towards a new approach for testing.
- Automatic test data generation is major issue in software testing problem. Most of the works reported in automatic test data generation but a new approach is required that can generate unique test data and that does not fall into local optima.
- Most of work with the hybridization of local search and heuristic techniques has done. There is limited work towards hybridization of metaheuristics algorithms in software testing. Hence more emphasis is required towards it.

From the survey of literature, it is concluded that metaheuristic techniques especially GA and PSO has become interesting preference for researchers to solve
testing problems. Development of heuristics and metaheuristics are still the major issues related to software testing which includes automatically test data generation to covers each and every statement. Therefore, in the present work, automatic test data generation problems with very good performance measures including generation of unique test data, covering each and every statement or 100 percent statement coverage have been considered. An attempt has been made to develop Hybrid algorithm that is based on combination of powers of two algorithms PSO and GA for solving test data generation problem of software testing which must be effective in generating test cases.