Webology (ISSN: 1735-188X)
Volume 19, Number 1, 2022

A Comprehensive Review Of Software Testing Methodologies Based On Search-Based Software Engineering

Dr. Mashael S Maashi
Department of Software Engineering, College of Computer and Information Sciences,
King Saud University, Riyadh, Saudi Arabia.
E-mail: mmaashi@ksu.edu.sa

Abstract

Model-based testing, structural testing, temporal testing, mutation testing, regression testing, exception testing, integration testing, interaction testing, and configuration testing are all applications of Search Based Software Engineering (SBSE). SBSE study attempts to use metaheuristic search techniques, genetic algorithms, and other methods to convert human-centered software engineering problems into machine-based search problems. This article examines the software testing future's potential possibilities by describing numerous Search Based Software Testing methodologies, analyzing research trends in this field, and investigating the software testing future's likely possibilities. This article examines Search Based Software Testing (SBST) as well as other modern computing disciplines that seamlessly overlap with SBST. The challenges that arise with the application of various approaches are also discussed in the study.

Keywords

Search based Software Engineering, Software Testing, Test Case Generation, Metaheuristic Search, Genetic Algorithm.

Introduction

Traditional software engineering optimization and testing methodologies have become a time-consuming process in recent years. The test data should be restructured as combinatorial optimization problems in order to speed up this procedure. This topic has been discussed and investigated in a variety of software development life cycle (SDLC) (Albalawi & Maashi, 2021) areas, which includes optimization of requirements, maintenance and refactoring of software code, optimization of test case, and debugging. (Bullnheimer et al., n.d.) define metaheuristic search to be a flexible parent process for obtaining solutions with high-quality in a fast and efficient manner. This is done by

monitoring and enhancing the subordinate heuristics operations. At each reiteration, this approach can handle single (incomplete or complete) solutions as well as sets of solutions. Low or high level techniques, a construction methodology, or simple local searches are examples of dependent heuristics. Metaheuristic algorithms are sometimes referred to as optimization algorithms or search based techniques whenever deployed to software engineering challenges, giving rise to Search Based Software Engineering (SBSE) (Harman & Jones, 2001).

Strategy that is one step removed from reality when trying to apply search-based optimization to physical engineering components: rather than the artifact itself, optimizing a simulation or representation of it. This also contributes to further potential inaccuracy and expense as the fitness calculated by this method is not the final product fitness.

In the context of testing, these approaches are known as Search-Based Software Testing (SBST) (Marculescu et al., 2012). The purpose of this paper is to provide an overview of recent trends in the use of search-based techniques to generate test sequences. A domain- specific software testing solution should ideally combine good software testing practices with domain knowledge and experience in application-specific quality assurance procedures and regulations. SBST has been proposed and validated for a variety of applications.

Highlights of this paper:

- Understanding SBSE.
- Definition of SBST and its background.
- Major approaches used to resolve the problem in SBST domain.
- Reviewing the Literature given in SBST.

The remaining of the paper is structured as follows. Section II includes an introduction of SBST definitions as well as a few of the most widely applied search algorithms in SBST test case development. This portion is again divided into two subsection as Evolutionary Test and Optimization Techniques which consist of five major search techniques. An overview of the method of conducting the survey, and some of the issues in implementation is given is Section III. Section 0 gives a synopsis of the paper and concludes the survey.

Background

The use of random or guided search techniques, such as hill climbing and genetic computations, to solve problems in software testing, verification, and approval is known as Search-Based Software Testing (SBST) (Ben Zayed & Maashi, 2021). Search-based methods are becoming more popular in programming testing, verification, and approval. They are especially useful in the generation of test data. Random search, local search (Mcminn et al., n.d.) (e.g. hill climbing, simulated annealing, and tabu search), evolutionary algorithms (Gupta et al., 2016) (e.g. genetic algorithms, evolution strategies, and genetic programming), ant colony optimization, and particle swarm optimization can be used to solve software testing problems as well as

confirmation and validation space. Other common modern software testing concepts include real-time testing, model-based testing, testing of service-oriented architectures, interface testing, test case prioritization, and data-driven test generation.

(Marculescu et al., 2012) has defined SBST as a cyclic process as shown in the Figure1 which consisting of following steps.

- **Initialization**- To initiate, a population of candidate solutions is generated. This is frequently done at random, but more advanced techniques can also be used.
- **Fitness Function**- Each candidate solution in the population is evaluated using a fitness function. The fitness function assigns a numerical value to each candidate solution and allows for the comparison of complex candidates.
- **Selection**- For the next generation, a subset of the original population of candidate solutions is chosen. The selection prioritizes candidate solutions with higher fitness, but other candidates may be chosen as well.
- **Population Generation**- The chosen candidates will serve as the foundation for the formation of a new generation. This is accomplished through the use of genetic operators. Mutation and crossover are two examples of such genetic operators. A candidate solution is mutated when a random modification is made to it. Crossover entailed combining existing candidate solutions to create new ones.

A test case or a set of test data could be a single candidate solution for SBST. The genetic algorithm will select test cases that dominate the quality criteria of the fitness function. Over several generations, the overall fitness of the candidate population is expected to improve.

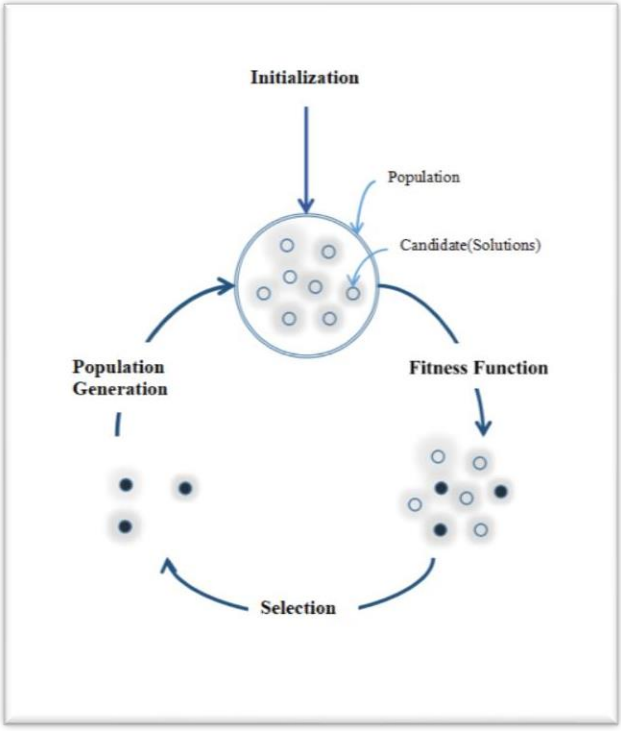

Figure 1 The basic idea of SBST using an example of a population-based genetic algorithm

1. Evolutionary Testing

In evolutionary testing, meta-heuristic search techniques are utilized to produce test cases. Evolutionary Testing (Paper & Studies, 2015) (Figure 2) is a subset of Search Based Testing in which an Evolutionary Algorithm is used to direct the query. The ultimate aim of this test has been transformed into an optimization complication. The search space is defined by the input domain of the test object. The test object scans the search space for test data that satisfies the specified test goal. A numerical representation of the test objective is required for this search. The objective functions that can be used to evaluate the test data generated are defined using this numerical representation. Depending on the test purpose, several heuristic functions for evaluating test data emerge. Because the software is not linear, converting test objectives to optimization problems usually results in convoluted, broken, and non-linear search spaces (if-statements, loops, and so on). As a result, neighbourhood search tactics (such as hill climbing) are out of the question. Meta-heuristic search methods include evolutionary algorithms, simulated annealing, and tabu search. Evolutionary Strategies (EAs) are fantastic optimization algorithms for software testing.

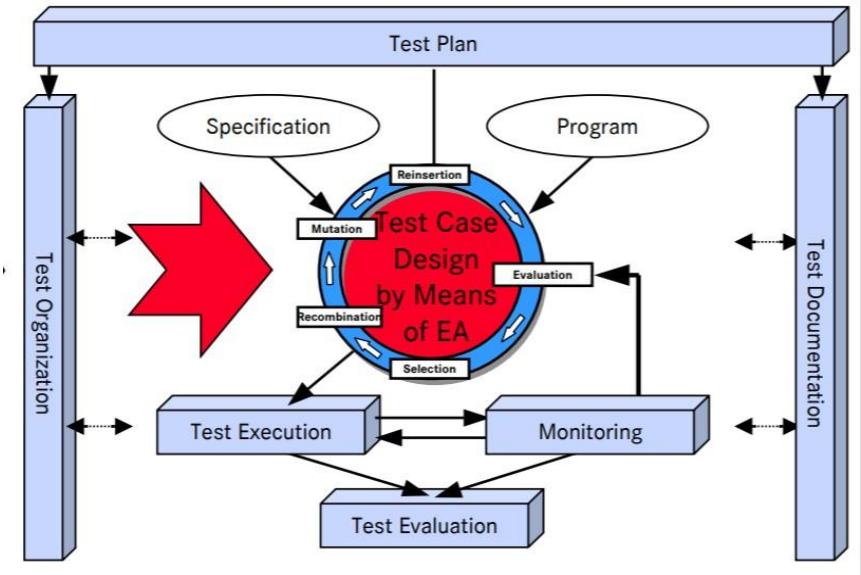

Figure 2 The structural overview of Evolutionary Testing

1. Optimization Techniques

In SBST, this research examines the most commonly used optimization techniques, such as Simulated annealing, Ant colony optimization, Genetic algorithm, Tabu search, and Particle swarm optimization.

i. Simulated Annealing

Local searches are performed using simulated annealing (SA). It takes a sample from the full domain and uses combination to change the arrangement in different ways. Simulated annealing

selects x1 as the value for the solution x, as well as the solution with the lowest cost (or objective) function E. The relative and desirability of different options are defined by cost functions. The cost function and the fitness function respectively, represent the processes of minimizing and maximizing the objective function.

ii. Tabu Search

The Tabu search (TS) algorithm is really a metaheuristic technique for solving combinatorial optimization issues like the travelling salesman problem (TSP). In this scenario, a local or neighbourhood search process is utilized to shift from solution s to solution s' in the vicinity of s until a stop requirement is reached. Tabu search enhances the neighbourhood design of each solution as the search advances by evaluating portions of the search space that might otherwise be unexplored by the local search technique.

iii. Genetic Algorithm

Genetic algorithms (GAs) (Maashi et al., 2020) are a sort of evolutionary algorithm that simulates scientific processes (such as SA), particularly the fertility and mutation processes in genetics, as well as natural selection. GAs analyse and prioritize the attributes of a variety of solutions using the fitness function (called "genes"). The solutions are measured in each generation to discover which are the "fittest." A subset of the population is chosen at each generation to reproduce the next generation of solutions in order to create diversity in the gene pool. To proceed, a crossover operator is applied to the selected parents to generate the child solution(s). The crossover operator is implemented differently depending on the algorithm, but in general, for the generation of an offspring, selections from each parent are moulded. The second step is mutation, which occurs after the child solution is created (s). Again, the implementation of the mutation is determined by the GA that was written. This mutation can be utilized to preserve diversity in solution selection while also avoiding convergence by injecting random modifications into the solution. After a subset of the child solutions has been mutated, the newly generated solutions will be stored into the gene pool. The algorithm now calculates the fitness of all new solutions and rearranges them in relation to the total set of solutions. A population size is often selected to guarantee that the weakest solutions are wiped out of the gene pool with each generation. This cycle is repeated until the stop condition is satisfied.

iv. Ant Colony Optimization

Ant colony optimization (ACO) (Dorigo & Blum, 2005) is a probability-based computation problem-solving algorithm that generates solutions by traversing a graph containing various system states. In software testing, ACO is used to generate test sequences. Stigmergy is defined as indirect communication between members of a population through contact with their surroundings, and ACO is founded on this principle. Stigmergy is an example of ant interaction during the foraging phase: ants communicate with each other indirectly by leaving pheromone trails on the ground, which influence other ants' decision-making. Similarly, the method takes a control flow graph as input and explores the nodes to discover the best pathways. For ACO implementation, the problem is represented as a graph with possible stops as nodes and possible paths as edges. The pheromone-depositing artificial ants are now travelling across these

pathways. A random path is picked if there is equal or no pheromone; otherwise, the road with the most "pheromone" is chosen because it is the shortest. The process is directed toward positive feedback as a result of its autocatalytic behaviour. Pheromones are laid and then disappear after a set length of time, which helps to avoid local optimal solution convergence.

Pusedocode for ACO algorithm:
Initialize: Trail
Do While (Criteria for halting the process were not met)
//Cycle Loop
Do Until(A TOUR is completed by each ant)
// Tour Loop
Local Trail Updates
End Do
Analyze Tours Global Trail
Updates
End Do

i. Particle Swarm Optimization

In the framework of the swarm intelligence paradigm, particle swarm optimization (PSO) is a relatively new optimization technique. It's a meta-heuristics technique to problem optimization that improves candidate solutions iteratively. The most typical application of PSO is to solve problems with solutions that can be described as a points in an n-dimensional space. A particle is the term used in PSO to describe a probable solution. Throughout this space, a number of particles are launched at random. The present position, velocity, and Pbest position of each particle are all known. Pbest is the most recent evaluation of a person's personal best position. Gbest, the worldwide best position earned by all of its members, is also included. It's a simple method that works in a variety of situations. Only a few steps make up the PSO algorithm, which are repeated until a terminating condition is fulfilled. The steps are given below:

1. Individuals' present position and velocity are used to initiate the population.
2. Individual particle fitness is evaluated (Pbest).
3. Keeping track of an individual's highest level of fitness (Gbest).
4. Velocity modification based on Pbest and Gbest location.
5. The particle position is being updated.

The first stage in PSO is to initialize the particle velocity and its current position. In this situation, the particle is in two-dimensional space. A function is used to calculate the particle's fitness value. Update the particle's x and y positions to reflect its personal best position if the particle's fitness is higher than its previous value. The particle's global best position is also updated when the value is larger than Gbest position. The process is repeated until the stop criteria have been met or the optimal option has been discovered.

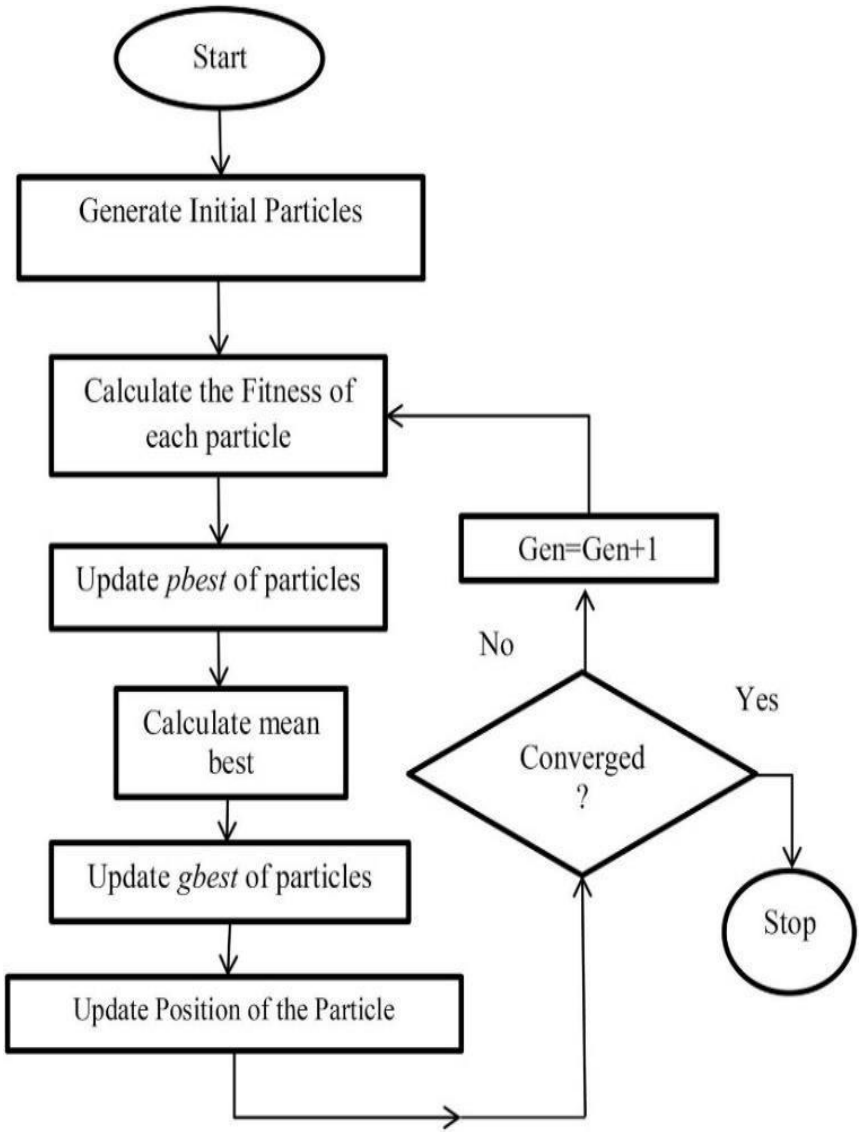

Figure 3 The flow chart Of Particle Swarm Optimization algorithm

Review on Related Works

Researchers' attention has inevitably been drawn to SBST as the scope of SBSE has grown. In recent years, there has been a spike in SBST, particularly in methods for test data generation that matches a set of criteria. (Mcminn et al., n.d.) provides an evaluation of SBST creation, demonstrating the use of search-based methodologies for Grey Box testing, White Box testing, Black Box testing, and nonfunctional characteristic validation. SBST can be implemented using both white box and black box methodologies.

Table 1 The use of five key metaheuristic methodologies in SBST domain research to solve diverse problems

	Author	Technique	Problems Addressed	Applied Models	Data Sets Used in Verification	Metrics/ Comparison techniques
1	(Waeselynck et al., 2007)	**Simulated Annealing (SA)**	Test Data Generation, safety critical property, testing, verification	SA, landscape concept	Two calendar & boiler problem, Quadratic assignment	Diameter, auto-correlation, generation rate of better solutions
2	(Lin & Yeh, 2001)	**Genetic Algorithm (GA)**	Deficiency of former test case generators	GA + Path Testing Algorithm	Test case to examine the target path	Target paths, fitness function
3	(Miller et al., 2006)		Automatic test data generation	GA + Test Data Generation	Six distinct programs	GADGET technique
4	(Gupta et al., 2016)		Optimized software testing	GA	Random numbers generated by code	Fitness function
5	(Babamir et al., n.d.)		Automated data collection for structural tests	GA	Min-max algorithms	Time order decline
6	(Pachauri, 2011)		Automation of the test case in java programs unit testing	GA, Java modeling language (JML)	Java framework JGAP, Java's software Soda-vending machine	Branch coverage
7	(Alshraideh et al., 2013)		Test data fulfilling branch criteria	GA	Six real objects of Jordan	Gaussian distribution, number creep

			for testing		university	& average

			PL/SQL programs		hospital information	coverage ratio
8	(Yu & Pang, 2012)		Optimizing software testing efficiency	GA, Control flow graph based methodology	-	Critical path clusters
9	(Ramasamy et al., 2021)	**Ant Colony Optimization (ACO)**	Rearrangement of the test cases within time limit	ACO + regression test prioritization	8 test cases containing 10 faults	Fault coverage capacity, Average percentage of fault detected
10	(Sharma et al., 2011)		Test sequences in state transitions optimal paths	ACO based tool efficient software coverage	State diagram of class management system state machine	Compared with conventional PPTACO
11	(Mao et al., 2015)		Generation of test data for structural testing	ACO, local transfer & global transfer	Eight significant programs	Average coverage, success rate, average generation time, ANOVA
12	(Díaz et al., 2008)	**Tabu search (TS)**	Automatic structural software testing	TS generator (TSGen)	Different C/C++ programs	Random generator
13	(Miranda, 2015)		Generating improved test-case sequences	TS, GA	Sample voter validation form	GA
14	(Windisch et al., 2007)	**Particle Swarm Optimization (PSO)**	Software testing	PSO, DaimlerChrysler test system	25 simple artificial & 13 complicated	Test case coverage, t-test

					industrial test objects	
15	(Ahmed et al., 2014)		GUI functional testing	Simplified swarm optimization	'Flight Reservation' case study	Pair-wise independent combinatorial, Test vector generator
16	(Wang & Liu, 2008)		Regression testing	Hybridized particle swarm optimization (HPSO), GA	Java's IDE	Execution time
17	(R.Girgis et al., 2015)		Automation of test paths generation	Genetical swarm optimization technique	15 C# small programs	PSO, GA for comparison

Issues with Implementation

Several challenges in SBST are guided by metaheuristic techniques, such as the production of automated test data and test case sequences. Until date, the true potential of any of these strategies for automatically generating test data for all sorts of software testing has not been fully identified. As a result, a more broad approach is required.

One of the other problems that has appeared is that recently suggested approaches including PSO and TS do not work autonomously; rather, majority of them are hybridized with traditional methods such as GA in various methods to tackle current software engineering problems. As a result, these modern approaches must seek an independent and distinct strategy to be implemented in the software engineering domain.

Third, different experts employed different performance comparison metrics to elucidate the efficacy of the suggested and enforced algorithm. C. Mao et al. (2015) chose success rate, average coverage, average time, and average generation as performance accounting indicators to compare their suggested approach with existing methods, whereas used branch coverage % as a strategy for comparison. As a result, standardized metrics should be used to evaluate and compare procedures so that a consistent result may be attained and followed.

Conclusion

In this article, we have decided to outline the numerous ways for automating the software testing phase of the SDLC. On the other hand, we've focused our efforts on evolutionary algorithms and search-based methods. The main goal is to see how search-based optimization methodologies might be utilized to automate software engineering solution evolution. When

compared to state-of-the-art Software Testing methodologies, SBST has a variety of advantages, including less work and increased authenticity.

The work gives a quick overview of what's been done so far, but it's not a complete picture of all that's been done so far. It will help greatly in exploring the difficulties that remain unsolved in the SBSE field especially in SBST and, as a result, resolving those problems by delivering productive, systematic, and feasible solutions. It can take advantage of the capabilities of various current computing technologies, such as evolutionary computing approaches like Genetic Algorithms, SA, and so on. This paper also shows how the research community is becoming more interested in this extremely promising area of software testing.

References

Ahmed, B.S., Sahib, M.A., & Potrus, M.Y. (2014). Generating combinatorial test cases using Simplified Swarm Optimization (SSO) algorithm for automated GUI functional testing. Engineering Science and Technology, an International Journal, 17(4), 218–226. https://doi.org/10.1016/j.jestch.2014.06.001

Albalawi, F.O., & Maashi, M.S. (2021). Selection and optimization of software development life cycles using a genetic algorithm. Intelligent Automation and Soft Computing, 28(1), 39–52. https://doi.org/10.32604/iasc.2021.015657

Alshraideh, M.A., Mahafzah, B.A., Salman, H.S.E., & Salah, I. (2013). Using Genetic Algorithm as Test Data Generator for Stored PL/SQL Program Units. Journal of Software Engineering and Applications, 06(02), 65–73. https://doi.org/10.4236/jsea.2013.62011

Babamir, F. S., Hatamizadeh, A., & Babamir, S. M. (n.d.). Application of Genetic Algorithm in Automatic. 545–552.

Ben Zayed, H.A., & Maashi, M.S. (2021). Optimizing the software testing problem using search-based software engineering techniques. Intelligent Automation and Soft Computing, 29(1), 307–318. https://doi.org/10.32604/iasc.2021.017239

Bullnheimer, B., Hartl, R. F., & Strauss, C. (n.d.). To the vehicle routing problem.

Díaz, E., Tuya, J., Blanco, R., & Javier Dolado, J. (2008). A tabu search algorithm for structural software testing. Computers and Operations Research, 35(10), 3052–3072. https://doi.org/10.1016/j.cor.2007.01.009

Dorigo, M., & Blum, C. (2005). Ant colony optimization theory: A survey. Theoretical Computer Science, 344(2–3), 243–278. https://doi.org/10.1016/j.tcs.2005.05.020

Gupta, P., Arora, I., & Saha, A. (2016). A review of applications of search based software engineering techniques in last decade. 5th International Conference on Reliability, Infocom Technologies and Optimization, ICRITO 2016: Trends and Future Directions, 978, 584–589. https://doi.org/10.1109/ICRITO.2016.7785022

Harman, M., & Jones, B.F. (2001). Search-based software engineering. Information and Software Technology, 43(14), 833–839. https://doi.org/10.1016/S0950-5849(01)00189-6

Lin, J.C., & Yeh, P.L. (2001). Automatic test data generation for path testing using GAs. Information Sciences, 131(1–4), 47–64. https://doi.org/10.1016/S0020-0255(00)00093-1

Maashi, M., Alwhibi, N., Alamr, F., Alzahrani, R., Alhamid, A., & Altawallah, N. (2020). Industrial Duct Fan Maintenance Predictive Approach Based on Random Forest, 177–184. https://doi.org/10.5121/csit.2020.100516

Mao, C., Xiao, L., Yu, X., & Chen, J. (2015). Adapting ant colony optimization to generate test data for software structural testing. Swarm and Evolutionary Computation, 20, 23–36.

https://doi.org/10.1016/j.swevo.2014.10.003

Marculescu, B., Feldt, R., & Torkar, R. (2012). A concept for an interactive search-based software testing system. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 7515 LNCS, 273–278. https://doi.org/10.1007/978-3-642-33119-0_21

Mcminn, P., Binkley, D., Mcminn, P., Mcminn, P., & Mcminn, P. (n.d.). Search-based so ware test data generation : a survey Related papers.

Miller, J., Reformat, M., & Zhang, H. (2006). Automatic test data generation using genetic algorithm and program dependence graphs. Information and Software Technology, 48(7), 586–605. https://doi.org/10.1016/j.infsof.2005.06.006

Miranda, T.B. (2015). Test-Case Optimization Using Genetic and Tabu Search Algorithm in Structural Testing, 4(5), 355–358.

Pachauri, A. (2011). Test Data Generation for Unit Testing of Java Programs Using JML and Genetic Algorithm. International Journal, 1(2), 11–20.

Paper, C., & Studies, E. (2015). Search based software testing: An emerging approach for automating the software testing phase of SDLC.

R.Girgis, M., S. Ghiduk, A., & H. Abd-Elkawy, E. (2015). Automatic Data Flow Test Paths Generation using the Genetical Swarm Optimization Technique. International Journal of Computer Applications, 116(22), 25–33. https://doi.org/10.5120/20469-2324

Ramasamy, J., Pundhir, S., Narayanan, S., Ramadass, S., Aswin, S., & Suresh, A. (2021). Deep learning for material synthesis and pose estimation material systems: A review. Materials Today: Proceedings. https://doi.org/10.1016/j.matpr.2021.04.234

Sharma, B., Girdhar, I., Taneja, M., Basia, P., Vadla, S., & Srivastava, P.R. (2011). Software coverage: A testing approach through ant colony optimization. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 7076 LNCS (Part 1), 618–625. https://doi.org/10.1007/978-3-642-27172-4_73

Waeselynck, H., Thévenod-Fosse, P., & Abdellatif-Kaddour, O. (2007). Simulated annealing applied to test generation: Landscape characterization and stopping criteria. Empirical Software Engineering, 12(1), 35–63. https://doi.org/10.1007/s10664-006-7551-5

Wang, D., & Liu, L. (2008). Hybrid particle swarm optimization for solving, 18(5), 1179–1183. Windisch, A., Wappler, S., & Wegener, J. (2007). Applying particle swarm optimization to software testing. Proceedings of GECCO 2007: Genetic and Evolutionary Computation

Conference, 1121–1128. https://doi.org/10.1145/1276958.1277178

Yu, B., & Pang, Z. (2012). Generating Test Data Based on Improved Uniform Design Strategy.

Physics Procedia, 25, 1245–1252. https://doi.org/10.1016/j.phpro.2012.03.228