International Journal of Management, IT & Engineering
Vol. 9 Issue 6, June 2019,
ISSN: 2249-0558 Impact Factor: 7.119
Journal Homepage: http://www.ijmra.us, Email: editorijmie@gmail.com
Double-Blind Peer Reviewed Refereed Open Access International Journal - Included in the International Serial Directories Indexed & Listed at: Ulrich's Periodicals Directory ©, U.S.A., Open J-Gage as well as in Cabell's Directories of Publishing Opportunities, U.S.A

A SURVEY ON USE OF SEARCH BASED OPTIMIZATION TECHNIQUES IN SOFTWARE ENGINEERING

Sanjiv Sharma [*]
S. A. M. Rizvi [**]
VineetSharma [***]

Abstract-

In recent years search based optimization techniques in software engineering has been a burgeoning interest among software engineering. The Search Based Optimization Techniques are used to shift problem of software optimization from human based search to machine based search, by using techniques like meta-heuristic search and evolutionary computation. The idea behind this paradigm is to mingle human creativity with computing machine's reliability. This article presents a survey on some good work already done in this field.

Keywords-search based software engineering; hill climbing; simulated annealing; evolutionary algorithms; testing

[*] Department of Computer Science and Engineering,,KIET Group of Institutions, Ghaziabad , India

[**] Department of Computer Science, Jamia Millia Islamia, Delhi, India

[***] Department of Computer Science and Engineering,KIET Group of Institutions, Ghaziabad , India

INTRODUCTION

The Search Based Optimization Techniques are used to shift problem of software optimization from human based search to machine based search, by using techniques like meta-heuristic search, evolutionary computation and operations research. These techniques try to combine human's creativity and machines reliability [1]. Search Based Software Engineering (SBSE) is the name of a field in which Search Based Optimization is applied to Software Engineering. In a search based problem optimal or near optimal solutions are hunted in a search space of candidate solutions. These solutions are guided by a fitness function that distinguishes between better and worse solutions [1]. The term SBSE was coined by Harman and Jones [2] in 2001, which discusses Search Based Optimization as a general approach to Software Engineering. SBSE has been used in many fields within the general area of Software Engineering, e.g., requirements, design and testing.

SBSE can be useful in finding out smallest set of test cases, best architecture of the system, set of requirements that optimizes development cost and customer satisfaction, optimal allocation of resources to software project and best sequence of refactoring steps [3]. The rest of the paper is organized as follows, Section 2 briefly discuss some of the search based optimization algorithms techniques. Section 3 discusses some applications of SBSE in software requirements and specifications. Section 4 presents some applications of SBSE in software design. Section 5 discusses use of SBSE in testing and last section is conclusion.

SEARCH BASED OPTIMIZATION ALGORITHMS

There are two important ingredients for application of search based optimization in software engineering problems. First is selection of the representation of the problem and second is definition of the fitness function. There are a lot of problems in software engineering that have software metrics associated with them. These metrics are good candidates for fitness function [3].With the help of these two ingredients it is possible to implement search based optimization algorithms. Every search based algorithm use different approach to find optimal or near to optimal solutions. Approach for finding of solution is based upon fitness function, because fitness function is used to compare candidate solutions.

A. *Hill Climbing Algorithm*:

This algorithm selects an initial candidate solution randomly. It then examines the other candidate solutions, which are present in the neighborhood of the initial solution. These candidate solutions are similar but differ in some aspect. If a neighboring candidate solution possesses better fitness value compared to current solution, the search moves to the new solution. Now algorithm explores neighborhood of this new solution for better solution, and so on, until there is no improvement on the current candidate solution. Solution obtained using this method is called locally optimal and may not represent globally optimal solution, so search is repeatedly restarted with different initial solution in order to find out best solution. Number of restart of algorithm is decided by computing resources and available time [4].

B. *Simulated Annealing Algorithm*

This algorithm was proposed by Kirkpatrick et al. [5], is variation of Hill Climbing algorithm that avoids the local maxima problem by allowing candidates solutions of poorer fitness value. The probability of acceptance p of an inferior solution changes during searching of solution, and is calculated as: $p=e^{-\delta/t}$, where δ is the difference value between fitness value between the current solution and the neighboring inferior solution being considered, and t is the control parameter known as the temperature. Temperature is cooled according to some cooling schedule. Due to high temperature at initial stage free movement is available is available in search space and it leads to lesser dependency on starting solution. As the search progresses, however, the temperature reduces, making moves to poorer solutions more and more unlikely. Eventually, freezing point is reached, and from this point on the search behaves identically to Hill Climbing. The name “Simulated Annealing” originates from the analogy of the technique with the physical process of annealing: the cooling of a material in a heat bath. When a solid material is heated past its melting point, and then cooled back into a solid state, the structural properties of the final material depends on the rate of cooling.

C. *Genetic Algorithms*

These algorithms come under the category of global searches due to considering many candidates solutions in the search space at once. The set of candidate solutions currently under consideration is called current population and each successive population considered is referred as a generation. These algorithms are inspired by Darwinian Evolution, here; each candidate

solution is represented as a vector of components referred to as individuals or chromosomes. Generally, a Genetic Algorithm uses a binary representation, i.e. candidate solutions are encoded as strings of 1s and 0s; however more natural representations to the problem may also be used, for example a list of floating point values. In Genetic Algorithms first generation is made up of randomly selected chromosomes. Each individual in the population is then evaluated for fitness. On the basis of fitness value, certain individuals are selected to go forward to the following stages of crossover, mutation and reinsertion into the next generation. In Holland's original Genetic Algorithm [6] fitness-proportionate selection was chosen. In this selection system, the expected number of times an individual is chosen for reproduction is proportionate to the individual's fitness in comparison with the rest of the population. Selection based on fitness value leads the search to prematurely converge. However Linear ranking [7] and Tournament selection [8] have been proposed to check these problems.

Once the set of parents has been selected, recombination takes places to form the next generation. Crossover is applied on individuals selected at random with a probability p_{cross} (referred to as crossover probability). After crossover, the offspring are inserted into the new population. If crossover does not take place, the parents are simply copied into the new population. After recombination, mutation is done, which is responsible for introducing genetic material into the search, in order to maintain diversification. This is generally achieved by flipping bits of the binary strings at some low probability rate p_{mute} (referred to as mutation probability), which is usually less than 0.01. The search is terminated when some stopping criterion has been met, for example when the number of generations has reached some pre-imposed limit.

I. REQUIREMENTS/SPECIFICATIONS

Requirements engineering is a elementary component of the Software Engineering process [9], to which SBSE has also been applied in order to optimize choices among requirements, the prioritization of requirements and the relationships among requirements and implementations. One main goal is to select near optimal subsets from all feasible requirements to satisfy the demands of the customers, while at the same time making sure that there are sufficient resources to undertake the selected tasks. In the NRP, the goal is to find the ideal set of requirements that balance customer requests, resource constraints, and requirement interdependencies is called Next Release Problem (NRP). NRP can be formulated as a search problem [10]. In this

formulation NRP problem is considered as a single objective optimization problem. In [10] applied a variety of techniques to a set of synthetic data to demonstrate the feasibility of SBSE for this problem. An iterative Genetic Algorithm proposed by Greer and Ruhe [11] for NRP. This approach balances the resources required for all releases; assessing and optimizing the extent to which the ordering conflicts with stakeholder priorities. Two objectives cost to the provider and estimated satisfaction rating for the customer is considered in [12].

Among various advantages of SBSE in requirement phase is robustness in volatile requirements [13], insight [12], requirements prioritization [13] and fairness in requirements assignment.
There are also some challenges in applying Search based Requirements Optimization, such as scalability, solution representation, fitness function definition, algorithm selection and requirement dependencies [14].

II. SOFTWARE DESIGN

The center of every software system is its architecture. Designing software architecture is a challenging task requiring much proficiency and knowledge of different design alternatives, as well as the ability to understand high-level requirements and piece them to detailed architectural decisions. Search-based approaches have been used in architecture level. In order to enhance and predict software quality search-based methods with some suitable fitness function are used in designing phase.

Amoui et al. [15] use the GA approach to improve the reusability of software by applying architectural design patterns to a UML model and to figure out the best sequence of transformations. Chromosomes are used for encoding of a sequence of transformations and their parameters. Each individual consists of several supergenes, each of which represents a single transformation. A supergene is a group of neighboring genes on a chromosome which are closely dependent and are often functionally related. Mutation randomly selects a supergene and mutates an arbitrary number of genes, inside that supergene and after this, check out its validity. If a transformed design contradicts with object-oriented concepts, for example, a cyclic inheritance, a zero fitness value is assigned to chromosome. Two versions of crossover are used. First one is a single-point crossover for supergene level, with a arbitrarily selected crossover point, which

swaps the supergenes beyond the crossover point. The second one crossover arbitrarily selects two supergenes from two parent chromosomes, and similarly applies single point crossover to the genes inside the supergenes. This combines the parameters of two successfully applied patterns. Quality of the transformed design is examined, as introduced in [16], by calculating its "distance from the main sequence" (D), which unite several object-oriented metrics by calculating abstract classes' ratio and coupling between classes, and measures the overall reusability of a system.

Bowman et al. [17] discuss the use of a multi-objective genetic algorithm (MOGA) in solving the class responsibility assignment problem with the objective of optimization of the class structure of a system through the placement of methods and attributes. The strength Pareto approach is used, which differs from a traditional GA by containing records of individuals from past populations. In paper each chromosome is represented as an integer vector. Each gene represents a method or an attribute in the system and the integer value in a gene represent the class to which the method or attribute in that locus belongs. A separate matrix is used for storing of dependency information between methods and attributes. Mutations are performed by simply changing the class value arbitrarily; One-point one traditional crossover is used. The fitness function is formed of five different values measuring cohesion and coupling: 1. method-attribute coupling, 2. method-generalization coupling, 3. method-method coupling, 4. cohesive interaction and 5. ratio of cohesive interaction. Selection is made with a binary-tournament selection where the fitter individual is selected 90% of the time.

Kessentini et al. [18] consider the transformation mechanism as a combinatorial optimization problem with the goal of finding out a better transformation for a given small set of variable examples. In order to achieve the goal authors combine transformation blocks extracted from examples to generate a model named model transformation as optimization by example (MOTOE). Model is based upon an adapted version of particle swarm optimization (PSO).

In this adapted PSO transformation solutions are represented as particles that exchange transformation blocks in order to converge towards an optimal transformation. Paper also discusses about two main advantages of MOTOE, it recommend a transformation without

deriving transformation rule first, and it can be operated independently from the source and target metamodels.

Räihä et al [19] proposed an approach for automation of synthesization of software architecture using genetic algorithms. This technique applies architectural pattern for mutation and quality metrics (modifiability and efficiency) for evaluation and produces a proposal for software architecture on the basis of functional requirement given as functional responsibilities in terms of a graph. The behavior of genetic synthesis process is analyzed with respect to the effect of dynamic mutation, quality improvement speed, and the effect of quality attribute prioritization. Research result shows that it is feasible to genetically synthesize architectures with high fitness value.

III. TESTING

Search-based software testing is the application of metaheuristic search techniques to generate software tests. The fitness function is transformation of test adequacy criterion and helps in finding out best solution in the search space. The application of metaheuristic search techniques for testing is useful, because exhaustive testing is infeasible for big size and complex softwares. Search-based software testing has been used across white-box (structural), black-box (functional) and grey-box (combination of structural and functional) testing.

Harman et al. [20] this paper presents a theoretical exploration of the global search technique embodied by Genetic Algorithms. After doing empirical study that compare the behavior of both global and local search based optimization on real world programs, it reveal that there exists of test data generation problem that suit each algorithm, thereby suggesting that a hybrid global-local search may be appropriate. The outcome of the study indicates that sophisticated search techniques, such as Evolutionary Testing can often be outperformed by far simpler search techniques. However, there exist test data generation scenarios for which the evolutionary approach is ideally suited. A further empirical study shows that in order to maximize coverage, Evolutionary Testing should be hybridized with the Hill Climbing approach.

Yano et al. [21] In this paper a new multi-objective implementation of the generalized extremal optimization (GEO) algorithm, named M-GEOvsl, is presented. It was developed to use as a test case generator to find transition paths from extended finite state machines (EFSM). This algorithm not only covers transition but also minimizes test lengths. M-GEOvsl can deal with variable length strings, and can generate solutions with different lengths. The steps of the algorithm are performed on general multi-objective problem in which the solution length is an element to be optimized. It must notice that although M-GEOvsl was developed in a specific context, it can in principle be applied to any multi-objective problem where the number of design variables is itself a variable of the problem. This paper has three main contributions, first is a dynamic approach used for generation of model-based test cases, in which the model is the artifact that is executed, instead of the implementation of system under test. Second is the transition paths, with the data that trigger them, in order to avoid infeasible path generation. Third is a multi-objective approach is proposed not only to cover the test purpose, but also to minimize the test case length. Fourth is dependence analysis is used to guide the search for solutions.

Assunção et al. [22] This paper presented MOCAITO (Multi-objective Optimization and Coupling-based Approach for the Integration and Test Order problem) approach for the integration and test order problem in varied software development contexts, where the units, components or classes can be components. The approach is instantiated in the object and aspect-oriented contexts, and evaluated with real systems and three algorithms: NSGA-II, SPEA2 and PAES. The algorithms are compared by using different number of objectives and four quality indicators. A dependency model is used to represent dependencies between units. In this paper ORD (Object Relation Diagram) model is used, and some most natural dependency relations between classes and between aspects and classes are considered. A cost model was also used after consideration of relevant information such as number of attributes, operations, and type and number of parameters of order cost. These two models are used as input to multi-objective optimization algorithms. The algorithms produce a set of beat solutions (good orders) that represent the best trade-off among the objectives. Now the tester selects an order according to the testing plans and environment, by using a priority rule. Results from the empirical evaluation of MOCAITO in OO (Object Oriented) and AO (Aspect Oriented) systems show that the three

compared evolutionary algorithms can efficiently solve the problem.. However, based on the analysis of the quality indicators Generational Distance (GD), Inverse Generational Distance (IGD), Coverage (C) and Euclidean Distance (ED) from an ideal solution, NSGA-II appears more suitable in most cases considering all systems, with second and fourth objectives. The algorithm PAES, shows better performance for more complex systems. SPEA2 presents the greatest execution time for all systems.

Boussaa et al. [23] In this paper, a NS (Novelty Search) algorithm based on statement-coverage criterion for the test data generation problem has been introduced. In this approach, algorithm explores the search space by considering diversity as the objective function and try to optimize it. Author selects test cases based on a novelty score showing how different they are compared to all other solutions evaluated so far rather than a fitness-based selection. Using the NS approach is clearly a divergent evolutionary technique, inspired by natural evolution's drive to novelty that directly rewards novel behaviors instead of progress towards a fixed objective.

Srivastava et al. [24] In this paper a method for optimizing software testing efficiency is presented. This objective is achieved by identifying the most critical path clusters in a program. Author has developed variable length Genetic Algorithms in order to optimize and select the software weighted path clusters based on criticality. Due to infeasibility of exhaustive software testing author developed a selective approach to testing by selecting on those parts that are most critical so that these paths can be tested first. In other word testing efficiency can be increased by identifying the most critical paths.

Langdon et al. [25] Author have represented mutation testing as a multiobjective search problem in which the goal is to search for higher order mutants that are difficult to kill and syntactically similar to the original program under test. The approach uses higher order mutation testing, but also consider traditional mutation testing since a first order mutant may be a special case of a higher order. This approach has been implemented using a combination of Genetic Programming (GP), Genetic Algorithms and Monte Carlo sampling. The results reveal that the higher order GP mutation testing approach is able to kill higher order mutant (complex faults) of a real program.

Shamshiri et al. [26] In this paper, an empirical study on working of two approaches; Genetic algorithm and Random Search is done is done by using EvoSuite (Automatic Test Suite Generation for Java) unit test suite generator and selecting 1,000 classes randomly from the SF110 corpus of open source projects. Results reveal that there is little difference between the coverage achieved by test suites generated by evolutionary search compared to test suite generated using random search. An exhaustive analysis reveals that the genetic algorithm covers more branches of the type where standard fitness functions provide guidance.

IV. CONCLUSION

This paper has surveyed applications of search based optimization techniques in different phases of software development. There are a various kind of search based optimization techniques available such as searches based on a heuristic, genetic algorithms and evolutionary computation. Some technique performs well in one scenario while other works in some other scenario. Quality of result produced depends directly upon algorithm chosen, representation of the problem in hand and most importantly on fitness function designed. In survey it has been observed that use of SBSE is least in requirement/specification and substantial in testing while design phase comes in between.

REFERENCES

[1] Harman, Mark, et al. "Search based software engineering: Techniques, taxonomy, tutorial." *Empirical software engineering and verification*. Springer Berlin Heidelberg, 2012. 1-59.

[2] Harman, Mark, and Bryan F. Jones. "Search-based software engineering."*Information and software Technology* 43.14 (2001): 833-839.

[3] Harman, Mark, S. Afshin Mansouri, and Yuanyuan Zhang. "Search based software engineering: A comprehensive analysis and review of trends techniques and applications." *Department of Computer Science, King's College London, Tech. Rep. TR-09-03* (2009).

[4] McMinn, Phil. "Search-based software test data generation: A survey."Software Testing Verification and Reliability 14.2 (2004): 105-156.

[5] S. Kirkpatrick, C. D. Gellat, and M. P. Vecchi. Optimization by simulated annealing. Science, 220(4598):671{680, 1983.

[6] J. H. Holland. Adaptation in Natural and Arti_cial Systems. University of Michigan Press, Ann Arbor, 1975.

[7] Darrell Whitley. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In J. D. Scha_er, editor, Proceedings of the International Conference on Genetic Algorithms, pages 116{121, San Mateo, California, USA, 1989. Morgan Kaufmann.

[8] K. Deb and D. Goldberg. A comparative analysis of selection schemes used in genetic algorithms. In Foundations of Genetic Algorithms, pages 69{93. Morgan Kaufmann, San Mateo, California, USA, 1991.

[9] Cheng, B., and J. Atlee. "From state of the art to the future of requirements engineering." Future of Software Engineering 2007 (2007).

[10] Bagnall, Anthony J., Victor J. Rayward-Smith, and Ian M. Whittley. "The next release problem." Information and software technology 43.14 (2001): 883-890.

[11] Des Greer and G¨unther Ruhe. Software release planning: an evolutionary and iterative approach. Information & Software Technology, 46(4):243–253, 2004.

[12] Yuanyuan Zhang, Mark Harman, and S. Afshin Mansouri. The multi-objective next release problem. In GECCO'07: Proceedings of the Genetic and Evolutionary Computation Conference, pages 1129–1136. ACM Press, 2007.

[13] Mark Harman, Stephen Swift, and Kiarash Mahdavi. An empirical study of the robustness of two module clustering fitness functions. In ACM Genetic and Evolutionary Computation Conference (GECCO 2005), Washington, D.C., USA, June 25-29 2005.

[14] Zhang, Yuanyuan, Anthony Finkelstein, and Mark Harman. "Search based requirements optimisation: Existing work and challenges." Requirements Engineering: Foundation for Software Quality. Springer Berlin Heidelberg, 2008. 88-94.

[15] Amoui, Mehdi, et al. "A genetic algorithm approach to design evolution using design pattern transformation." International Journal of Information Technology and Intelligent Computing 1.2 (2006): 235-244.

[16] Martin, Robert C. "Design principles and design patterns." Object Mentor 1 (2000): 34.

[17] Bowman, Michael, Lionel C. Briand, and Yvan Labiche. "Solving the class responsibility assignment problem in object-oriented analysis with multi-objective genetic algorithms." Software Engineering, IEEE Transactions on36.6 (2010): 817-837.

[18] Kessentini, Marouane, Houari Sahraoui, and Mounir Boukadoum. "Model transformation as an optimization problem." Model Driven Engineering Languages and Systems. Springer Berlin Heidelberg, 2008. 159-173.

[19] Räihä, Outi, Kai Koskimies, and Erkki Mäkinen. "Genetic synthesis of software architecture." Simulated Evolution and Learning. Springer Berlin Heidelberg, 2008. 565-574.

[20] Harman, Mark, and Phil McMinn. "A theoretical and empirical study of search-based testing: Local, global, and hybrid search." Software Engineering, IEEE Transactions on 36.2 (2010): 226-247.

[21] Yano, Thaise, Eliane Martins, and Fabiano Luis De Sousa. "A multi-objective evolutionary algorithm to obtain test cases with variable lengths."Proceedings of the 13th annual conference on Genetic and evolutionary computation. ACM, 2011.

[22] Assunção, Wesley Klewerton Guez, et al. "A multi-objective optimization approach for the integration and test order problem." Information Sciences267 (2014): 119-139.

[23] Boussaa, Mohamed, et al. "A Novelty Search-based Test Data Generator for Object-oriented Programs." Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference. ACM, 2015.

[24] Srivastava, Praveen Ranjan, and Tai-hoon Kim. "Application of genetic algorithm in software testing." International Journal of software Engineering and its Applications 3.4 (2009): 87-96.

[25] Langdon, William B., Mark Harman, and Yue Jia. "Efficient multi-objective higher order mutation testing with genetic programming." Journal of systems and Software 83.12 (2010): 2416-2430.

[26] Shamshiri, Sina, et al. "Random or Genetic Algorithm Search for Object-Oriented Test Suite Generation?." Proceedings of the 2015 on Genetic and Evolutionary Computation Conference. ACM, 2015.