Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 6.199

IJCSMC, Vol. 8, Issue. 6, June 2019, pg.124 – 127

A Review on Search Based Software Engineering

Mrs. Divya K V
Assistant Professor, Department of Information Science and Engineering,
New Horizon College of Engineering, Bangalore
divya.k.vasudevan@gmail.com

Abstract-- The plan of Search Based Software Engineering (SBSE) research is to shift software engineering problems from human-based search to machine-based search, by means of the of the techniques from the metaheuristic search and evolutionary computation paradigms. The design is to utilize humans' ingenuity and machines' tenacity and reliability, rather than requiring humans to perform the more monotonous, error prone aspects of the engineering process. SBSE has been applied to problems all over the Software Engineering lifecycle, from requirements and project planning to maintenance and re-engineering. This paper provides a review on SBSE in Software development life cycle (SDLC) and various algorithms used in SBSE

Keywords: SBSE, metaheuristic search, SDLC

I. INTRODUCTION

The phrase SBSE was first used in 2001 by Harman and Jones [1]. SBSE converts a software engineering problem into a computational search problem that can be tackled with a metaheuristic. This involves defining a search space, or the set of possible solutions. Search-based software engineering is relevant to almost all phases of the software development process. SBSE is important in software engineering because it provides a mode to attack hard, extremely constrained problems that involve many objectives using an automated approach. It is quite easy to apply because representations and fitness functions are voluntarily available in software engineering.

SBSE seeks to reformulate Software Engineering problems as search-based optimization problems. This is not to be confused with textual or hyper textual searching. Rather, for Search Based Software Engineering, a search problem is one in which optimal or near optimal solutions are sought in a search space of candidate solutions, guided by a fitness function that distinguishes between better and worse solutions. This paper aims to provide a survey of SBSE, A wide range of different optimization and search techniques can and have been used. The most widely used are local search, simulated annealing, genetic algorithms and genetic programming.

In other engineering disciplines, such as mechanical, materials, chemical, electric, and electronic engineering, search-based optimization has been applied for many years. It is only recently that software engineering has started to catch up with this trend. There is every reason to think that this growing interest will continue. In some ways, the

advent of SBSE is merely a reflection of the evolution of software development into a mature and fully fledged engineering discipline. On the other hand, software engineering, more than any other engineering discipline, is redolent with search based optimization application potential. The very nature of software makes it even better suited to search based optimization than traditional engineering artifacts. In traditional engineering optimization, the work of art to be optimized is often simulated. This is typically necessary precisely because the artifact to be optimized is a physical entity. Search-based optimization requires repeated fitness computation, which is impractical if a physical solution has to be constructed for each fitness evaluation. Engineers seeking to apply search-based optimization to traditional physical engineering artifacts therefore have to content themselves with an approach that is one step removed from reality; optimizing not the artifact itself, but some simulation or representation of it. The fitness thus computed is not the fitness of the final product, resulting in a supplementary layer of potential inaccuracy and cost.

II. SBSE IN SDLC

A. SBSE for Software Testing

SBSE has been applied to many software testing applications, including structural testing, model-based testing, mutation testing, temporal testing, exception testing, regression testing, integration testing, configuration, and interaction testing [2].However, in literature, there has been little work in search-based approaches to statistical testing. Empirical results that support the claim that the distributions of test cases they derive using SBSE outperform the fault finding abilities of traditional structural testing techniques. Studies address the problem of stress testing also.

B.SBSE for Software Design

Software design is one of the recent areas of growing SBSE research interest. Software design presents the engineer with a naturally complex design space in which many competing and conflicting objectives must be optimized and balances between different concerns must be found. SBSE research focused on fully automated approaches to fitness computation in which the human input to the overall design of good fitness functions is crucial to the success of SBSE [2]. Also, there has been little previous work on the direct involvement of the human software engineer in the search itself. Software design uses SBSE techniques to guide the decision maker in their design process. Using multi-objective optimization and SBSE approach can seek to find a balance between several competing objectives. These different objectives may be in competition with one another. In this situation, researchers have found Pareto optimal approaches to be very attractive.

C.SBSE for Software Modeling and Prediction

Modeling and prediction are also topics for which SBSE is well adapted because it can cater to multiple, potentially conflicting goals and constraints and can be used to interpolate a best fit to a set of data, using fit as a fitness function. Optimizing software quality models can improve predictive capability, employing genetic programming to build optimized models from multiple data sets. SBSE can be used to help predict the performance characteristics of component-based system assemblies. The approach uses SBSE to build models using genetic programming, from which a behavioral model is formed. A combination of dynamic and static analysis is used to generate the required input for genetic programming.

III. SEARCH TECHNIQUES

There are numerous different metaheuristic algorithms available to use in the SBSE field. These methods are used to automate search-based problems through gradual quality increases [3].Random search is used as a benchmark for most search-based metaheuristic algorithms to compare against. Although most metaheuristics use a non-deterministic approach to making choices, the choice must be assessed for validity and a fitness function is used to evaluate whether the search should continue from that point or backtrack.

A. Hill climbing (HC)

Hill climbing (HC) is a type of local search algorithm. With the HC approach, a random starting point is chosen in the solution, and the algorithm begins from that point. A change is then made, and the fitness function is used to compare the two solutions. The one with the highest perceived "quality" becomes the new optimum solution and the algorithm continues in this way. Over time, the quality of the solution is improved as less optimal changes are discarded and better solutions are chosen. Eventually, an optimal or sub-optimal solution is reached with the same functionality but a better structure. This is considered a fast algorithm in relation to the other metaheuristic choices but, as with other local search algorithms, it has the risk of being restricted to local optima. The algorithm may "peak" at a less optimal solution (akin to reaching a peak after climbing a hill). There are two main types of HC search algorithm that differ in one aspect [3]. First-ascent HC is the simpler version of the algorithm, whereas steepest-ascent HC has a slightly more sophisticated search method and is a superior choice for quality. Other variations are stochastic HC (neighbors are chosen at random and compared) or random-restart HC (algorithm is restarted at different points to explore the search space and improve the local optima reached). HC is one of the more common search algorithms used in SBSE, and has similarities to other search techniques.

B. Simulated annealing (SA)

Simulated annealing (SA) is a modification of the local search algorithm, used to address the problem of being trapped with a locally optimum solution [4]. In SA, the basic method is the same as the HC algorithm. The metaheuristic checks stochastically between different variations of a solution and decides between them with a fitness function until it reaches a higher quality. The variation is that it introduces a "cooling factor" to overcome the disadvantage of local optima in the HC approach. The cooling factor adds an extra heuristic by stating the probability that the algorithm will choose a solution that is *less* optimal than the current iteration. While this may seem unintuitive, it allows the process to explore different areas of the search space, giving extra options for optimization that would otherwise be unavailable. This probability is initially high, giving the search the ability to experiment with different options and choose the most desirable neighborhood in which to optimize. This is then generally decreased gradually until it is negligible. The probability given by the cooling factor is normally linked to a "temperature" value that is used to simulate the speed in which the algorithm "cools". Although the SA process may come up with a better solution compared to the HC process, HC is a lot more reliable as the SA process may struggle to settle on a solution.

C. Genetic algorithms (GAs)

Genetic algorithms (GAs) are a class of evolutionary algorithms (EAs) that, much like SA, mimic a process used elsewhere in science, namely the reproduction and mutation processes in genetics and natural selection. GAs use a fitness function to measure the quality among a number of different solutions (known as "genes") and prioritize them [6]. At each generation (i.e. each iteration of the search), the genes are measured to determine which are the "fittest". Each generation, in order to introduce variation into the gene pool, a proportion of the population is selected and used to breed the new generation of solutions. With this selection, two steps are used to create the new generation. First, a crossover operator is used to create the child solution(s) from the parents selected. The algorithm itself determines exactly how the crossover operator works, but generally, selections are taken from each parent and spliced together to form a child. Once the child solution(s) have been created, the second step is mutation. Again, the mutation implementation depends on the GA written. The mutation is used to provide random changes in the solutions to maintain variation in the selection of solutions and prevent convergence. After mutation is applied to a selection of the child solutions, the newly created solutions are inserted back into the gene pool. At this point the algorithm calculates the fitness of any new solutions and reorders them in relation to the overall set. Generally, a population size is specified, and this ensures that the weakest solutions are weeded out of the gene pool each generation. This process is repeated until a termination condition is reached.

D. Multi-objective evolutionary algorithms

When refactoring a software project, as with other areas of software engineering, there are likely numerous conflicting objectives to address and optimize. A multi-objective algorithm can be used to consider the objectives independently instead of having to combine them into one overarching objective to improve. There are numerous EAs available that are used for multi-objective problems, known as multi-objective evolutionary algorithms (MOEAs). The downside to using multi-objective algorithms for software refactoring over the mono-objective metaheuristic algorithms is that the extra processing needed to consider the various objectives can cause an increase in the time needed to generate a set of solutions. Another issue is that when a MOEA generates a population of solutions, the "best solution" is up to the interpretation of the user. Whereas a single-objective EA can rank the final population of solutions by a single fitness value, there may be numerous possible choices in the MOEA population depending on which objective fitness is more important. On the other hand, this gives the user multiple options depending on their desire or the situation.

IV. CONCLUSION

The study indicates that other areas of activities in software engineering are starting to receive significant attention: Requirements, Project Management, Design, Maintenance and Reverse Engineering. The paper shows the comparison between the search techniques Genetic algorithm, simulated annealing and hill climbing. Genetic Algorithm is proved to be the best algorithm because it focuses on solutions and fitness. They can be easily implemented and understandable in comparison to hill climbing and simulated annealing. Genetic Algorithms are best for combinatorial problems.

REFERENCES

[1] Mark Harman, S. Afshin Mansouri and Yuanyuan Zhang. *"Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications*", April 9, 2009

[2] Mark Harman and Afshin Mansouri, *"Search Based Software Engineering"*, Introduction to the special issue of the IEEE Transactions on Software Engineering. Vol. 36, No. 6, November/December 2010.

[3] Arushi Jain, Aman Jatain *"Search Based Software Engineering Techniques",* International Journal of Computer Applications (0975 – 8887) Innovations in Computing and Information Technology (Cognition 2015)

[4] Himmat Singh, Aman Jatain, Hitesh Kumar *Sharma "A Review on Search Based Software Engineering",* International Journal Of Research in Information Technology.Vol.2, Issue 4,April 2014,pg:116-122.

[5] Mark Harman Baryan F. Jones, *"Search Based Software Engineering"* Information and Software Technology 43(2001)833-839.

[6] Francisco Javier Rodriguez-Diaz, Carlos GarciaMartinez, and Manuel Lozano, *"A GA-Based Multiple Simulated Annealing"* 2010, IEEE.

[7] Rob A. Rutenbar, *"Simulated Annealing Algorithms: An overview",* January 1989.

[8] E-G.Talbi, P.Muntean, *"Hill climbing, Simulated Annealing and Genetic Algorithms: A comparative Study and Application to the Mapping Problem*", 1993 IEEE.

[9] Abdel Salam Sayyad, Hany Ammar. *"Pareto-Optimal Search based Software Engineering",* 2013 IEEE.

[10] Wasif Afzal, Richard Torkar, and Robert Feldt. *"A systematic review of search-based testing for nonfunctional system properties"*. Information and Software Technology, 51(6):957–976, 2009.

[11] Giulio Antoniol, Stefan Gueorguiev, and Mark Harman. *"Software project planning for robustness and completion time in the presence of uncertainty using multi objective search based software engineering. In ACM Genetic and Evolutionary"* Computation COnference (GECCO 2009), pages 1673–1680, Montreal, Canada, 8th – 12th July 2009.

[12] J. E. Baker. *"Reducing bias and inefficiency in the selection algorithm".* In Proceedings of the 2nd International Conference on Genetic Algorithms and their Application, Hillsdale, New Jersey, USA, 1987. Lawrence Erlbaum Associates.

[13] Andrea Arcuri. "*It does matter how you normalise the branch distance in search based software testing".* In Proceedings of the International Conference on Software Testing, Verification and Validation, pages 205–214. IEEE, 2010.

[14] Edmund Burke and Graham Kendall. *"Search Methodologies. Introductory tutorials in optimization and decision support techniques".* Springer, 2005.

