Test Suite Optimization using Nature Based Metaheuristic Approach

A dissertation submitted in the partial fulfillment for the award of Degree of

Master of Technology

In

Software Engineering

by

Vishal Gupta (2K14/SWE/20)

Under the guidance of

Prof (Dr.) Daya Gupta (Former HOD)

Department of Computer Engineering, DTU

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

BAWANA ROAD, DELHI

DECLARATION

I hereby want to declare that the thesis entitled "**Test Suite Optimization using Nature Based Metaheuristic Approach"** which is being submitted to the **Delhi Technological University**, in partial fulfillment of the requirements for the award of degree in **Master of Technology in Software Engineering** is an authentic work carried out by me. The material contained in this thesis has not been submitted to any institution or university for the award of any degree.

_____

Vishal Gupta

Department of Computer Engineering

Delhi Technological University,

Delhi.

CERTIFICATE

Delhi Technological University
(Government of Delhi NCR)
Bhawana Road, New Delhi-42

This is to certify that the thesis entitled **"Test Suite Optimization using Nature Based Metaheuristic Approach"** done by **Vishal Gupta** (2K14/SWE/20) for the partial fulfillment of the requirements for the award of degree of **Master of Technology** Degree in **Software Engineering** from **Department of Computer Engineering**, Delhi Technological University, New Delhi is an authentic work carried out by him under my guidance.

Project Guide:
Prof. Daya Gupta
Professor and Former Head of Department
Department of Computer Engineering
Delhi Technological University, Delhi

ACKNOWLEDGEMENT

I take this opportunity to express my deep sense of gratitude and respect towards my guide **Prof. Daya Gupta (Former Head of Department) Department of Computer Engineering.**

I am very much indebted to her for her generosity, expertise and guidance I have received from her while working on this project. Without her support and timely guidance the completion of the project would have seemed a far –fetched dream. In this respect I find myself lucky to have my guide. She have guided not only with the subject matter, but also taught the proper style and techniques of documentation and presentation. I would also like to take this opportunity to present my sincere regards to **Ms. Shruti Jaiswal**, Research Scholar, DTU for extending her support and valuable Guidance.

Besides my guides, I would like to thank entire teaching and non-teaching staff in the Department of Computer Engineering, DTU for all their help during my tenure at DTU. Kudos to all my friends at DTU for thought provoking discussion and making stay very pleasant.

Vishal Gupta
M.Tech Software Engineering
2K14/SWE/20

Abstract

Regression testing is a type of software testing that intends to validate modified software and it confirms that modifications made to the software have no adverse side effects. The goal of regression testing is to validate the modified software. Due to the resource and time constraints, it becomes necessary to develop techniques to minimize existing test suites by eliminating redundant test cases and prioritizing them.

Running all test cases in an existing test suite can consume an inordinate amount of time and resources. Thus, it is necessary to select the minimum set of test cases from existing test suite with the ability to cover all the faults in minimum execution time.

When analyzing large test suits, redundancies are identified in test cases, hence, it is necessary to reduce these suites, in order to fit the available resources, without severely compromising the coverage of the test adequacy criterion being observed. Test case prioritization techniques intend to arrange test cases of a test suite in a way, with the goal of maximizing some objective function. There are various classical techniques of test case prioritization. In this research we proposed a nature inspired meta-heuristic approach for test prioritization.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1 General Idea

Real World is filled with various hard and complex problems. One such complex problem is an optimization problem. Optimization has been an active area of research for several decades. It deals with finding the best solution from the set of all feasible solutions. Optimized solutions are hard to find so there are no deterministic algorithms that can find exact solution in polynomial time. Several techniques have been proposed to solve these hard problems. Computational Intelligence (CI) is one of the techniques to solve these problems. Algorithms based on this technique are nature inspired computational techniques to address real world's complex problems for which traditional methodologies can be useless because of the complexities of the process or because of the random and stochastic nature of problems. CI algorithms have previously been applied to various domains like fault diagnosis, robotics and control, virus detection and anomaly identification etc.

In large domain of applications of intelligence techniques we are interested in finding the application of meta-heuristic algorithms to the domain of software testing. Meta-heuristic algorithms are guided by the concept of exploration and exploitation. The exploration phase refers to the process of investigating the promising area of the search space as wide as possible whereas, exploitation refers to the local searching capability around the solutions obtain in exploration phase. Reason for the popularity of meta-heuristic algorithm stands on these four pillars: simplicity, flexibility, derivation-free mechanism and local optima avoidance. Metaheuristic are fairly simple and are inspired by very simple concepts. Inspirations are generally from the physical phenomenon, animal behaviours and evolutionary concepts. Flexibility refers to the applicability of meta-heuristic to different problem domains without any special changes in the structure of the algorithm. Majority of meta-heuristics operates on derivation free mechanism in contrast to gradient based optimization approaches since these algorithms optimizes problems stochastically. Because of this randomization meta-heuristic have

the superior abilities to avoid the local optima as compared to traditional optimization techniques. Metaheuristic algorithms like Ant Colony Optimization(ACO), Biogeography based Optimization(BBO), Grey Wolf Optimizer(GWO) and Cuckoo Search(CS) have already been applied to the various domains like data mining and fraud detection but are not limited to computer science domain only and are also fairly well known among other fields like biomedical, social science and military applications.

Area of software testing has also been lit by the application of meta-heuristic algorithms. Our research has been inclined towards the application of meta-heuristic algorithms. Developing software is difficult but testing and building confidence to use that software is in fact more difficult because if software is complex then so does its test suite. Researches have shown that more time and cost is consumed testing and maintaining the product rather than developing the product. So creation of test cases and thus order of their execution must be done sincerely. In the past Hla et. al [1] prioritise the test cases based on the altered part using PSO. Hybrid of genetic algorithm and Metaheuristic like PSO has been proposed to choose minimum set of test cases that covers all possible faults [2].

Regression Testing is the verification process to determine that previous functionalities of the software remain the same after software is changed and the previously tested software has not been introduced to any new errors. It is a maintenance phase activity. Complete retesting is desired after maintenance but re-executing each test case is not feasible during maintenance because of time and cost constraints when the software is large. Hence in order to achieve complete coverage testing within stipulated amount of time test case prioritization and minimization is required.

1.2 Motivation

Test suite is as set of test cases that are designed to be used to test some specified behaviour of a software program. Test Case is a document which has a set of test data along with preconditions, expected results and post-conditions in order to verify the compliance against a specific requirement.

A single test case can be part of multiple test suites. This property of test cases results into execution of same test cases again and again. Moreover during regression testing executing all test cases are not possible, so there is need of test suite prioritization and minimization. A

selective retesting technique reduces the cost of testing a modified program by reusing the existing tests.

Test suite minimization is a process of reducing those test cases whose existence does not improve the efficiency of test suite. Only those test cases are selected which are not redundant and provide exhaustive testing. So removal of these cases would not hurt the fault tolerance characteristics of the system [3].

Test suite prioritization is a process of assigning ranks or priorities to test cases based on some objective functions like fault tolerance or execution time [4]. Test prioritization provides a more efficient and structured way to execute test cases and thus find faults within stipulated time.

Goals of prioritization can be stated as follows:

- To increase the rate of fault detection of test suites i.e. the likelihood of revealing faults earlier in a run of regression tests using prioritize suites.
- To increase the coverage of code in the system under test at a faster rate, allowing a code coverage criterion to be met earlier in the test process.
- To increase confidence in the reliability of the system under test at faster rate.
- To increase the likelihood of revealing faults related to specific code changes earlier in the testing process.

There have been various techniques applied to achieve the above goals some of these are stated in next section.

1.3 Related Work

Test Case prioritization has been a hot topic of research from past few years. Test Case prioritization and minimization has been improved by various techniques in previous researches.

Li et. al [5] applied greedy algorithm, genetic algorithm and hill climbing based on the principle that those test cases are selected which covers maximum testing requirements. Genetic algorithm depends upon migration operator and genetic algorithm tends to stuck at local optima.

Tallum [6] proposed a new greedy heuristic algorithm for choosing a subset of test cases T' from test suite T which have the ability to cover all requirements as covered by T. Greedy Approach sometime stuck at local maxima.

In [7] a nature inspired technique is used to modify the process of test minimization and test prioritization.

Singh et. al [8] proposed Ant Colony optimization which works on the real life behaviour of the ants. Singh explains how randomness characteristic in ACO helps in exploration of optimal test case order. ACO uses single fitness function that helps in optimal solution identification. Major fall back of ACO is dependency upon initial factors.

Yoo and Harman [9] applied evolutionary techniques for structural test case selection. Evolutionary algorithms generate solution using operators like crossover, selection, and mutation. Nature Inspired algorithms have no particular interest in discontinuity and differentiability whereas traditional algorithm like hill climbing are influenced by correlation between values like time and fault covered to compare test cases.

Bob Simon [10] introduced Biogeography Based Optimization which has been applied for test suite prioritization, because it proves to be useful in other hard problems.In [40] they try to map BBO hybrid SA to prioritize test cases which shows better results than many other meta-heuristic algorithms.

In [12] Ritika Nagar et. al tries to leverage the property of obligate broad parasitism of some cuckoo species to solve the prioritization problem of test cases but cuckoo search depends on the probability of detecting alien eggs and involves biased generation of new cuckoo eggs.

Seyedali Mirjalili [11] proposes a new meta-heuristic called Grey Wolf Optimizer (GWO) inspired by grey wolves. The GWO algorithm mimics the leadership hierarchy and hunting mechanism of grey wolves in nature. Four types of grey wolves which formulate the hierarchy are alpha, delta, omega and beta. In addition three main steps of hunting, searching for prey, encircling prey and attacking prey are implemented to stimulate the wolves' behaviour. Grey wolf proves to be comparatively efficient than many other algorithms to solve NP Hard problems which requires selecting the optimal solution from the combinatorial search space. This research will explore use of grey wolf meta-heuristic for test case prioritization.

1.4 Problem Statement

The Test Case Prioritization Problem*:*

Given: *T, a test suite; PT, the set of permutations of T; f, a function from PT to the real numbers.*

Problem: *Find T' € PT such that for all T'' where* (*T'' € PT*) (*T'' ≠ T'*) [*f (T'') ≥ f (T)*]

PT represents the set of all possible prioritization of T, f is function that applied to any such ordering yields an fitness value for that ordering

Grey Wolf optimizer (GWO) has shown promising and more than satisfactory results in the various domains like Thermal power systems [14], feature subset selection [13], time forecasting [15], vehicle routing [16] and optimizing key values in cryptographic algorithms [17]. GWO has been successfully applied to solve binary [13] and multi-objective problems [18].

As stated previously various techniques have been applied to solve test prioritization problem. Unlike traditional techniques Grey Wolf Optimizer does not incorporate gradient functions which were used in various statistical algorithms which were used in [19]. Focus of our research will be on the exploration of GWO algorithm in the domain of test case prioritization and minimization.

Adapt GWO for test suite prioritization and minimization and evaluate its performance with other nature inspired meta-heuristics.

1.5 Objective and Scope

GWO mimics the social hierarchy and hunting behaviour of grey wolves. Social hierarchy constitutes the division of the pack of wolves into categories namely alpha, delta, gamma and omega. Alpha wolves are the most dominant ones and everyone else follows the dominant wolves. Similar concept is used to prioritize and minimise the test cases. GWO uses fault detection capability of test suite as cost function to compare two test suites. Major advantage which put GWO ahead of other nature based meta-heuristic is number of initial parameters required to initialise the algorithm is just two which are: pack size and maximum generation limit.

We empirically analysed the performance of GWO and thus compare it with other meta-heuristic algorithms under two categories namely Ant Colony Optimization (ACO), Cuckoo Search (CS), Biogeography based optimization (BBO) which falls under population based meta-heuristics and Simulated Annealing (SA) which is categorised under single solution based meta-heuristics based on the following parameters: Number of redundant test cases identified, Number of initial parameters required, Number of test cases identified, Convergence rate of algorithm, Variance in the final result on multiple runs.

We have chosen MATLAB programming language to implement GWO, ACO, BBO, SA on Jmeter dataset, which is obtained after compiling Jmeter separately for Jmeter's Junit test cases.

The scope of this thesis can be summarised as:

- Adapt GWO for test case prioritization and minimization
- Empirical study of GWO for test case prioritization on Jmeter as dataset
- Comparing the results of GWO, BBO, ACO, SA and Cuckoo.

1.6 Organization of the thesis

The rest of this paper is organized as follows.

Chapter 2 provides details about the past research done on improving regression test suite optimization.

Chapter 3 provide details about Earlier approaches like Ant Colony optimization, Biogeography based optimization, simulated annealing and cuckoo search in solving Test Suite prioritization(TSP) problem.

Chapter 4 provides general details about grey wolf in algorithm in optimization problems

Chapter 5 shows the procedure as to how to adapt grey wolf optimizer in solving test prioritization

Chapter 6 provides information about the dataset, simulation environment and Results and its analysis

Chapter 7 is about conclusion and future work and references, where all the research articles which contributed to this research are listed.

Chapter 2

LITERAURE SURVEY

In this chapter we introduce the process for test suite prioritization, which is categorized as a NP Hard problem in software engineering domain, explained various approaches in software testing and thus elaborate key factors driving the heuristic algorithms. Following it we present a literature survey on the existing work done in the field of test suite prioritization. This section discuss about the past algorithms and techniques applied for test suite prioritization.

2.1 Test Suite Prioritization

Test suite is as set of test cases that are designed to be used to test some specified behaviour of a software program. It is a container of test cases designed with a problem and its scope in mind. Test Cases are grouped together based on different modules or faults in the software under test. A test suite reports the execution status of the test cases and it can be in any of the states viz. Active, In-process and completed.

A single test case can be part of any number of test suites. Formation of the test suites is followed by formulation of test plans showing the execution cycle of the test suite. A set of large number of test cases formed a single test suite.

A test suite consists of tests which may be based on functional or non-functional requirements.

Test suite is said to be executed successfully if it completes the testing process by comparing each test case with the corresponding post condition. In order to define the completion of the testing process we need to define the testing completion criterion.

Few test completion criterion can be listed as follows:

- A predefined pre-decided amount of testing coverage has been attained
- No fatal defects are present in system
- Only few known lesser priority faults that don't affect the usage of the product in general a lot are present in system.

Significance of Test completion criterion are:

- Exit or completion criteria is essential to stop the testing process
- Certain level of quality of project has to be assured before quitting the testing process..
- Defining the amount of resources involved in testing cycle can be useful after successful completion of completion criteria

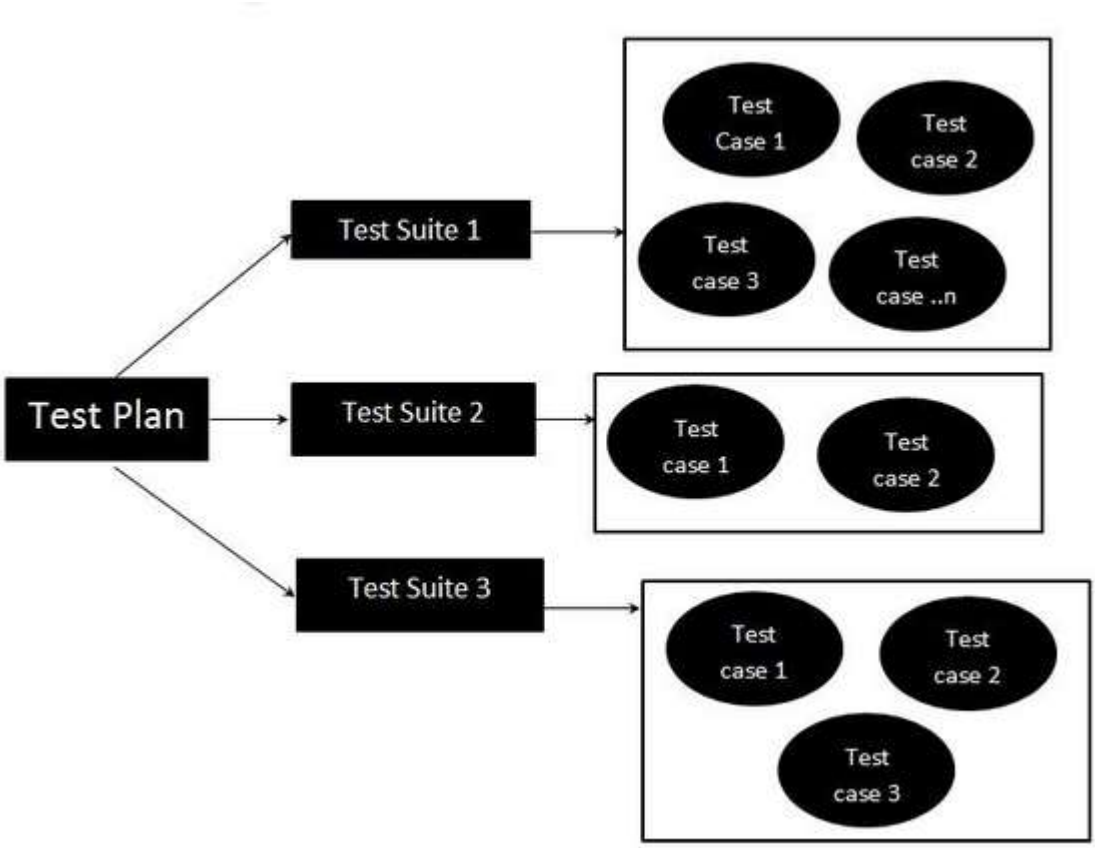


Figure 1 Test Suite Design Diagram [55]

2.1.1 Approaches for Software Testing

Testing can be done by multiple ways few approaches can be given as follows:

- Consultative approach: Approaches which may ask the users or non-testers of the system to tell you what to test.
- Model based approaches: In this approach the test process of the software are based on UML diagrams like State Chart Diagram, Activity diagram etc. System must behave

according to tests that are predicted by the model, and then the system is considered to be working.

- Risk based approaches: Test cases are designed based on the risk determined and analyzed during the requirement elicitation and specification phase.
- Methodical traditional sequential approaches: These approaches are based on some model like waterfall model, which are sequential in nature. Methodical test strategies stick to a pre-planned, systematized approach gathered from various concepts developed in-house and from the outside
- Heuristic Approaches: These are intelligence based approaches that are based on some statistics, fitness functions, probabilities, or operators like crossover, selection and mutation. Since they do not make any assumption about the nature of problem, they are well suited in solving the problem because of this adaption property. In this thesis we focus on heuristic approaches to solve Test suite prioritization problem.

2.2 Key Operations in Heuristics

Test suite prioritization problem is an NP-Hard problem. Problem with NP-Hard is that they can only be solved in exponential time by using traditional statistical algorithms which are deterministic. In order to make Test Suite Optimization problem solvable in polynomial time we employee meta-heuristics algorithms like Ant Colony Optimization, Particle Swarm Optimization etc. which are based on randomness.

The process of solving test suite prioritization problem using these meta-heuristics involves following key operations[1, 2, 8, 19]:

- **Understanding the problem:** First, Problem must be analyzed and the problem domain must be understood thoroughly. In our case the test cases of software are to be prioritized and minimized, so understanding the urge to priorities and previous techniques applied to do so along with their drawbacks is necessary.

- **Preparation of parametric table and required data set:** After understanding the problem and deciding the algorithm to be used a parametric table or data set is collected. In this parametric table, we identify all the attributes that are directly or indirectly influencing the problem under test. In our problem the parametric table is a matrix, with

rows, representing the test cases and columns define the faults in the software covered by those test cases.

- **Formulate the fitness functions and identify constraints:** Fitness function defines a formula formulating a numeric value which gives the goodness of the solution based on the value of attributes possessed by data set row. The fitness function is formulated based on constraints derived from the problem domain. In our problem we decide to use average percentage of fault detected as a fitness function.

- **Type of Optimization Operation (maximization or minimization):** During the process of optimization we employee the fitness function to be operated on every population over the generation and each generation either maximize or minimize the fitness function. The number of generations and the change in fitness function value are trivial parameters to prevent algorithm to struck at saddle point, local minima or to prevent over fitting and under fitting.

- **Optimization Process and Evaluation:** This phase involves the execution of algorithm selected on the formulated data set; we either use statistical technique or heuristic technique for solving the optimization problem. Since meta-heuristics involves randomization their results are evaluated using some known metrics like maximum, minimum, mean square (MS), root means square (RMS), variance etc. followed by the analysis of the result.

In this work of test suite prioritization and minimization we have used average percentage of fault detection (APFD), by a test case ordering, as the fitness function and complete test coverage, redundant test cases identified as the constraints against which we minimize and prioritize the test cases in the test suite.

In order to understand the work done in thesis following key points needs to be understood clearly and necessarily [55]:

- Regression Test minimization is an important problem under testing. It involves identification of reusable and redundant test cases which is basic principle of test case minimization.
- Formation of test artifacts like test plans, test strategy and test suite must be performed in parallel with requirement, design and coding phase so that accurate and efficient test suites can be designed which should reflect the need of the customer.
- Minimization enables the software tester to include efficient and useable test cases. It makes test suite robust by omitting out redundant test cases from test suites.
- Choosing suitable metric and thus analyzing the results helps in defining the effectiveness of the regression test suite ordering. The declaration of priorities of the test case based on APFD metric states the effectiveness of algorithm used in deriving those results.

2.3 Related Work

Utch, Chu, Elbaum, Malishevsky and Harold [20] [21] empirically investigate six test case prioritization techniques based on code coverage and fault exposing potential showing that prioritising test suite exposes more faults. According to Sanchic [22] test case prioritization is useful to reduce the quality assurance cost.

Elbaum el. al [23] proves that how significant is the order of execution of test case in quickly identifying the faults. Improved rate of fault detection is able to provide faster feedback on the system under test and help the tester in locating and correcting faults earlier than otherwise possible. In [24] he introduce a metric which incorporates varying test case and fault cost to predict the usability and efficiency of test case

Sudhir Mohapatra and Srinivas [25] applied evolutionary algorithm which prioritise the test case based on the code coverage of the test cases rather than their fault detection capability. In their implementation they applied genetic algorithm empowered with crossover and mutation operator. Major limitation of their technique involves that research was based on prioritization based on code coverage rather than fault detection rate.

In [26] Mirarab and Talivildari uses Baysian network approach in the prioritization technique which incorporates source code changes, software fault proneness and test coverage data. Complexity of baysian network and need of lot of training data limits their applicability in large softwares. Further, linear regression is traditional technique which tends to stop at local optima and provides single solution. Metaheuristic algorithms overcome both these problems hence they comes out to be better algorithm.

Kaur, Arvinder et. al [27] proposes the application of Bee Colony Algorithm to provide solution to the problem of regression testing. In their proposed work, bee colony algorithm was used for attaining maximum fault coverage in minimum execution time. In [28] they try to do the same thing using a hybrid of particle swarm optimization (PSO) and genetic algorithm (GA). Their technique merges the exploration process of particle swarm optimization and exploitation was obtained by using mutation operator of genetic algorithm. As usual meta-heuristic algorithms prove to be extraordinary while performing test suite prioritization because of their randomness.

Ali Hadar, Aftab et. al [29] proposed a on the fly test suite optimization technique using fuzzy logic which incorporates multiple objectives while prioritization. Fuzzy logic stands on the idea of generation of set of rules that affects the label attributes but this is not always the case in test suite prioritization, as in regression testing we need to achieve maximum coverage based on constraints like faults covered and execution time. In fact there can be test cases which do not increase the efficiency of the system and test cases at all. So no rule will be generated for those test cases. However the idea of test suite prioritization can be beneficial for regression testing domain.

Ryan Carlson el. al [30] implemented a prioritization technique that incorporates a clustering approach to help test case prioritization techniques on real data set. Their work proves that the test case prioritization technique which incorporates clustering improves the effectiveness of prioritization technique.

Greedy Technique was proposed to prioritize the test cases which select and orders the test cases based on the maximum number of requirements covered by the test case and should have the minimum overlap with other selected test cases. The idea was tested on Siemen suite and space program. The research was focused on the idea of minimum overlapping but overlapping does not affects the testing issues. The main idea of regression testing is to have minimum number of

test cases based on complete coverage relying on fault coverage and execution time not on overlapping criteria.

Singh et. al [8] proposed Ant Colony optimization which works on the real life behaviour of the ants. Singh explains how randomness characteristic in ACO helps in exploration of optimal test case order. ACO uses single fitness function that helps in optimal solution identification.

Vivekanandan el. al [31] improves the regression testing by incorporating ant colony optimization with dynamic dependency injection. The idea was to prepare a set of test cases which possess the potential to detect any bug that creeps in after the system starts operating in real environment with actual values of the variables. The algorithm was modified by dynamic injection dependency on the best route identified by ant colony algorithm.

Sahar Tahvali et. al [32] proposed an approach for prioritizing test cases based on multiple criteria by using Analytic Hierarchy Process (AHP).They applied AHP in a fuzzy environment so that criteria value can be specified using fuzzy variables when precise quantified values are not available. Approach was also applied for testing non-functional requirements in the system.

Kumar Harish et. al [33] proposed a hierarchical approach for test case prioritization which were based on requirements coved by test cases mapped from the requirement specification document. Their approach analysed and assigned values to each requirement based on the comprehensive set of twelve factors. The prioritize requirements were mapped on the highly relevant modules and then prioritize the set of test cases.

In [37] support vector machine classifier was used to identify infeasible test cases in the test suite. Test cases are termed as infeasible if they terminate prematurely and are responsible for wastage of software resources. So in order to reduce wastage in term of computation cost, memory usage, processing time and execution resources we need to remove infeasible test cases. The method of induced grammar was used to make SVM learn infeasible test cases since method of supervised learning is not feasible for software test prioritization.

Luay Tahat et. al [34] presented and evaluated two model based methods viz. selective method and a dependency based method of test prioritization using the state based model of the system under test assuming that modification were made on both system under test and the model of the system. Information about already executed test cases was used to prioritize. Research was based

on the fact that execution of model is inexpensive comparatively to the system under test hence the overhead of test prioritization is very less.

Srivastava et. al. [35] proposed an approach based on meta-heuristic firefly algorithm to generate optimal test paths. They modify firefly algorithm by defining appropriate objective function and introducing guidance matrix in traversing the graph. Attractiveness of firefly is directly proportional to the objective function value of a firefly.

Gregg Rothermal et. al [36] proposed a regression test selection technique by constructing control flow graph for a procedure and its modified version and then select tests that execute changed code from the original test suite. The set of tests selected by this technique includes tests from the original test suites that can expose fault in program. However this algorithm may select some tests that may not expose any fault.

In [38] research introduced an artificial intelligent concept of case based reasoning which tries to minimize the size of the tests and time while preserving the fault detection. But major limitation with this technique includes uncontrollable cost issue as same that of testing.

Jung-Min Kin et. al [42] proposed addition of memory to the regression testing because memory less prioritization implicitly assumes that local choices ensure adequate long performance which may not be necessarily the case with test prioritization. Instead, they collaborated historic execution data with prioritization and conduct experiment to assess its impact on the long run performance of regression testing. Research provide the trade-offs which should be considered while using prioritization technique over a series of software release.

In the next chapter we try to adapt various meta-heuristic algorithms like ACO, BBO and Cuckoo search in solving test case prioritization and minimization problem. We also juxtaposes simulated annealing, a trajectory based single population Metaheuristic in solving test case prioritization, so as to bring in to light the clear comparison. We will then, discuss in detail about adapting GWO for test case prioritization.

Chapter 3

Review of Metaheuristic Algorithm for Test Suite Prioritization

This chapter describes the algorithm of Ant Colony Optimization (ACO), Biogeography Based Optimization (BBO), Cuckoo Search (CS) and Simulated Annealing (SA) for test case prioritization.

Metaheuristic algorithms are problem independent procedures that provide a framework to develop heuristic optimization algorithm. Metaheuristic algorithms involve trade-off of randomization and local search. Randomizations provide reliant way to deviate from the local search to the search on global level. Thereby, almost every meta-heuristic algorithm is suitable for global optimization. Metaheuristic Algorithms have the capability to achieve solution with limited information and computational capacity [56]. Among the found solution it is expected that some are of optimal acceptable quality though there is no guarantee.

For difficult optimization problems like test suite prioritization finite amount of time to solve the problem increases exponentially. Deterministic heuristics may fail to find good solution in reasonable amount of time. Here meta-heuristic algorithms perform extraordinary well. Moreover Metaheuristic intends to learn from the solutions and fitness functions used are not based on differentiability and smoothness of the curves.

The meta-heuristics taken into the account are Ant Colony Optimization (ACO), Biogeography Based Optimization (BBO), Cuckoo Search (CS), Grey Wolf Optimizer (GWO) and Simulated Annealing (SA).

We divide the applied algorithms in two categories viz. earlier approach and proposed approach based on the time frames in which algorithms are proposed.

The category of earlier approaches includes Ant Colony Optimization (ACO), Biogeography Based Optimization (BBO) and Cuckoo Search (CS) which are population based meta-heuristic and Simulated Annealing which is single solution meta-heuristic algorithm.

Proposed approach includes test case prioritization using grey wolf optimizer which is a population based Metaheuristic.

Test suite prioritization techniques are illustrated on real world test suite of Jmeter. Faults covered by these test cases are categorised into 11 classes viz. Resource not found, File cannot be created, Function not found, Invalid variable or Parameter, Invalid Query, Alias, Invalid file access, In valid results, Out of range, End of file and divide by zero. Thus test case formulated to be used as test data for the implemented algorithms. Based on fault detection value and time to implement metric named average percentage of fault detected (APFD) proposed by Elbaum [23] is calculated for each test case ordering to rank the test cases. The ranking thus obtained is used to assist in test case minimization, details of which are explained in later chapters.

3.1 Ant Colony Optimization

Ant Colony Optimization (ACO) is a population based meta-heuristic algorithm used to solve NP Hard optimization problems. Ant colony optimization inspired from the mimicking behavior of some ant species. These ants deposit some pheromone on the ground in order to mark favorable path that should be followed by other members. Ant colony optimization exploits a similar mechanism for solving optimization problems. ACO forms the numerical information from pheromone content that will provide solution to the problem. It involves formation of acyclic graph based on continuous communication between ants via pheromone trails. Pheromone values are used and updated by the ACO algorithm while searching. During ACO's search ants try to adapt them using pheromones showing learning.

Ants initially wander randomly, and upon finding food return to their colony while leaving behind a chemical named pheromone as trail while returning. If other ants roaming randomly find such a path with high pheromone deposit they stop travelling randomly and start following the trail and reinforce it while returning. Mean of communication between ants are sound, touch and pheromone. The use of pheromones as chemical signals is most developed in ants. When better found source is identified by some ant then ant stop marking trails while returning and the start following the new trail helping ants to adapt the changing environment. Successful trails are followed by more ants resulting into more reinforcement and hence a better shorter path is identified. Pheromone evaporates with time reducing its attraction strength. Longer paths turned into greater evaporation so shorter paths at last are inevitable. Evaporation of pheromone helps in avoiding convergence to local solutions. The idea of the ant colony algorithm is to mimic this behavior of ants with virtual simulated agents walking over the graph, representing the problem to be solved.

An ant in ACO signifies a simple computation agent, whose movement results in optimization of the problem. ACO arrives at the solution iteratively. In every iteration, there is change in ordering from, say x to y, where y is closer to the optimal solution. The movement of any ant from state x to state y depends on:

- Attractiveness Level (xy) : priori probability of that move based on some heuristic.
- Trail Level $\tau(xy)$: posteriori probability is the indication of desirability of the move.

During each generation in test suite prioritization, we try to update APFD of each test case, with simultaneously keeping older values, forming a trail and new values are responsible for attractiveness. Hence the movement of a test case from APFD x to APFD y is stated through following probability formula:

$$p^k(xy) = \frac{t^\alpha(xy)\eta^\beta(xy)}{\sum t^\alpha(xy)\eta^\beta(xy)}$$

$t^\alpha(xy)$ is amount of pheromone (APFD value) in the trail array whereas $\eta^\beta(xy)$ is amount of pheromone that determine the attractiveness quotient. α And β are parameter values for controlling the influence.

3.1.1 Applications of Ant Colony Optimization

ACO has found it application in domains like network routing for routing packets, image processing for edge detection [43], bioinformatics in which it is used in DNA sequencing and DNA matching, vehicle routing problems. Moreover, ACO has been used in solving traditional color problem, travelling salesman problem, 3-SAT problem and partition problem.

ACO has already been implemented in various domain of software engineering like software quality estimation, software requirements prioritization, software project time line design and test suite prioritization as it can be used to solve problems in less time with lesser complexity [44].

3.1.2 Algorithm of ACO for Test Suite Prioritization

ACO has been applied in the process of generation of test suites for state based software testing. A state based dynamic graph of the software under test was assessed using group of ants to generate optimal test suites ordering [45]. Prioritization of test cases/suites is done so that

maximum faults can be recovered in minimum time. Since nature inspired algorithm are efficient in time constraint optimization problems, ACO serves better in prioritization of test cases. Below, we have provided an algorithm of ACO for solving Test Suite Optimization (TSO) problem. In solving Test Prioritization, the word pheromone is synonymous to value of average percentage of fault detected metric.

Formulate a parametric table, with rows defining the test cases and columns defining the faults covered by those test cases for Jmeter's data set.

Each Ant represents a different test case ordering, which can be a potential solution.

Objective Function: Average Percentage of fault detection

Initial pheromone value: Average fault percentage detection considering two test cases i and j only for the entry P_{ij} where P is Pheromone matrix

Terminating Condition: Number of iterations

I. Define initial pheromone value
II. Place each ant on initial state with empty memory.
III. While not the Number of iterations as described
    a. For each Ant:
        a. Until all test cases are covered

            Ants move from its initial empty ordering to selecting the prospective next test case in the ordering based on the probability values depends upon pheromone level which is defined by average percentage of fault detection.

    b. Calculate fitness of ant using Objective Function
    c. Check for the best order obtained
    d. If the transition of ants showed improvement from previously obtained best order, update the global best obtained
    e. Update Trails :

        For each ant:

            Evaporate a fixed amount of pheromone from each ant

    f. Update the pheromone table

IV. End While

V. Best Ant obtained represents the prioritized order of test cases.

VI. Identify the redundant test cases using the following function:

a. From least ranked test case to highest ranked test case:

i. If faults covered by test case is already covered by higher ranked test cases

Remove the test case

ii. Else

Add test case to front of test cases to be executed

VII. Final list contained the prioritized and minimized set of test cases

3.2 Biogeography Based Optimization

Biogeography Based optimization (BBO) is a population bio-inspired meta-heuristic algorithm based on evolutionary algorithm which optimizes the function by the virtue of randomization iteratively. BBO is a swarm intelligence algorithm which has the ability to solve NP Hard problem. Unlike traditional algorithms BBO does not uses gradients of function which makes it indifferent to complex differential equations. BBO maintains population of solutions and thus creating new solutions by combining the existing ones using operators like speciation, migration and extinction.

Biogeography can be understood as study of distribution of species and in geographical space. Modern biogeography is extensively observed on evolution on species on islands because of their easy observations [46]. This biogeography is thus mathematically model on the foundation of speciation of new species, migration of special between islands and extinction of species from the islands. By speciation we mean evolution of new species, migration involves movement of the species between the islands and extinction removes the species from the habitat. Island is any habitat which is geographically separated from other habitats. Emigration of species occur as species can float, swim or ride he wind to reach neighboring habitat. Areas with highly favorable conditions are said to have high habitat suitability index. Emigration from the habitat does not mean that the species completely disappears from its original habitat only a few representatives of species emigrate, so an emigrating species remains present on its original habitat while at the same time migrating to a neighboring habitat. Habitats with high HSI support more species and

habitat with low HSI can support few species. Biogeography is nature's way of distributing species, and is analogous to general problem solutions.

The features variable correlating with HIS that determine the nature of habitat is known as Suitability Index Variable (SIV). Each habitat represents a candidate solution to an optimization problem. Habitats with high value for Habitat suitability Index not only have a high emigration rate, but they also have a low immigration rate because of the high number of inhabitants in that habitat. Habitats with high value of habitat suitability index results in death of immigrant due to high competition for resources whereas habitat with low value of habitat suitability index have high value of immigration rate as a result large species will migrate to these islands. Low population keeps the competition very less but there will be increase in diversity being a good migration candidate. When there will be high immigration on the island having low HSI, its HSI value will increase.

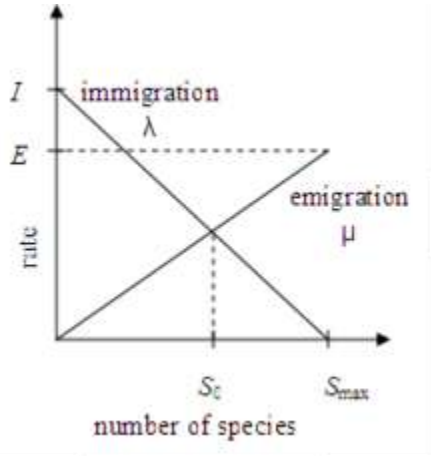


Figure The variation of immigration and emigration probabilities in BBO

λ here means immigration probability and δ means emigration probability. For any species k, its immigration rate is α_k and its emigration rate is β_k.

$$\alpha(k) = I\left(1 - \frac{k}{Smax}\right)$$

$$\beta(k) = \left(\frac{Ek}{Smax}\right)$$

The prominent factors that affect the working of the algorithms are:

- The migration operator
- The mutation operator
- The probabilities of immigration and emigration

Migration operator works on the fitness values of the individuals based on the curve formed from immigration and emigration probabilities affecting the working of Biogeography Based Optimization algorithm. By this we are taking into account species from different habitat.

Mutation operator brings in diversity in the population and provides assurance of convergence with development of Pareto fronts and not just one optimal solution. Mutation operator is often used in an optional manner, in cases like when random probability is less than mutation probability or when the archive array is not showing diversity in the population. The mutation operator is inversely proportional to the number of species and probability of the habitat. If individual has lower value for probability of speciation then it has higher value of probability of mutation and hence it can be considered as a better individual. Similarly, the individual with higher values for speciation will have lower mutation probability rates and hence it will not easily mutate with other individuals.

3.2.1 Application of Biogeography based Optimization

The application of biogeography to engineering is similar to what has been achieved in the past few decades with other computational intelligence algorithms like genetic algorithms (GAs), neural networks, fuzzy logic, particle swarm optimization (PSO), and many others because it relies on same evolutionary meta-heuristic. In fact Yang et. al. [48] shows that BBO is superior than GA, PSO and ABC. Problems like power flow [47], case based reasoning for retrieving ground water possibility, satellite image classification etc. were solved efficiently using Biogeography based optimization.

Like PSO, GA and ACO biogeography based optimization has already been implemented in various domain of software engineering like software quality estimation, software requirements prioritization, software project time line design and test suite prioritization as it can be used to solve problems in less time with lesser complexity [40]. Since nature inspired algorithm are efficient in time constraint optimization problems, BBO performed excellently in prioritization

of test cases. Below, we have provided an algorithm of BBO for solving Test Suite Optimization (TSO) problem. In solving TSO, the word habitat is synonymous to single test case ordering.

3.2.2 Algorithm of BBO for Test Prioritization

Formulate a parametric table, with rows defining the test cases and columns defining the faults covered by those test cases for Jmeter's data set.

Each habitat represents a different test case ordering, which can be a potential solution.

$\{data\}_k$ represents in Boolean table, showing faults covered by each test case, k= {1, 2, N}

Terminating Condition: Number of iterations

I. Define an Elite array for selection best individual in each generation and set its size ratio nKeep of the size of population
II. Immigration Rate= 1- Migration Rate
III. Mutation probability (σ) <- values [0, 1]
IV. Define an initial random population
V. While (not the maximum generation limit achieved)
    a. Define a Random number
    b. For each $\{data\}_k$, set the emigration probability δ_k proportional to fitness of $\{data\}_k$, $\delta \in [0, 1]$
    c. For each $\{data\}_k$, set the immigration probability λ_k , λ_k <- 1- δ_k
    d. Set {temp}<-{data}
    e. For each individual {temp}
        i. Use immigration probability to identify the individual which has be emigrated
        ii. Selection of immigrant is based on roulette wheel
        iii. Perform mutation with probability pMutation
    f. Keep nKeep number of good habitat along with Population-nKeep*Population new population for next iteration
VI. End While
VII. Best Habitat obtained represents the prioritized order of test cases
VIII. Identify the redundant test cases using the following function:

a. From least ranked test case to highest ranked test case:
    i. If faults covered by test case is already covered by higher ranked test cases
        Remove the test case
    ii. Else
        Add test case to front of test cases to be executed

IX. Final list contained the prioritized and minimized set of test cases

Algorithm of Mutation modified for Test Prioritization

i. For i=1: number_of_habitats(N)
ii. Do
    a. If random < mutation_probability
    b. Replace X(i) with a random Sutability Index Variable
    c. End if
iii. End for

Algorithm of Migration modified for Test Prioritization

i. For i=1: number of individual (N)
ii. Do
    a. Select _individual < probability_immigration
    b. If random < probability_immigration
    c. Select_source_individual ‘j’ using Roulete Wheel Mechanism
    d. Random selection of a variable ‘k’ to select X(j,k) from X(j).
    e. Replacing X(i,k) with X(j,k).
    f. End if
iii. End for

3.3 Simulated Annealing

Classification of meta-heuristic can be done in numerous ways. One way is to classify them as: population-based and trajectory-based meta-heuristic. For example, genetic algorithms are population-based as they use a set of strings; similar is the particle swarm optimization (PSO) which uses multiple agents or particles.

Simulated annealing on the other hand uses a single agent or solution which moves through the design space or search space in a piecewise style. A better move or solution is always accepted while a not-so-good move can be accepted with a certain probability considering that this not-so-good trajectory can also lead to the global optimum. The steps or moves trace a trajectory in the search space and this trajectory gives us the global optima.

Simulated annealing (SA) is developed in 1983as optimization technique inspired by the annealing process of metals. It is a trajectory-based search algorithm starting with an initial guess solution at a high temperature, and gradually cooling down the system. A move or new solution is accepted if it is better; otherwise, it is accepted with a probability, which makes it possible for the system to escape any local optima. It is then expected that if the system is cooled down slowly enough, the global optimal solution can he reached. Unlike the gradient-based methods and other deterministic search methods which have the disadvantage of being trapped at local minima, the main advantage of simulated annealing is its ability to avoid being trapped in local minima

Simulated annealing (SA) is one of the earliest and yet most popular meta-heuristic algorithm. It mimics the annealing process in material processing when a metal cools and freezes into a crystalline state with the minimum energy and larger crystal size so as to reduce the defects in metallic structures. The annealing process involves the careful control of temperature and its cooling rate, often called annealing schedule. This complete method is known as Metropolis method and markov's process is followed until equilibrium is achieved. It has been proved that simulated annealing will converge to its global optimality if enough randomness is used in combination with very slow cooling.

Simulated annealing is a search algorithm via Markov chain. However using markov chains may not be very efficient. Practically, it is usually beneficial to use multiple Markov chains in parallel to increase the overall efficiency of algorithm. In fact, the algorithms such as particle swarm

optimization can he viewed as multiple interacting Markov chains, though such theoretical analysis remains almost intractable.

In every aspect, a simple random walk can also be considered as a Markov chain. Briefly speaking, a random variable α is a Markov process if the transition probability, from state $\alpha_t = S_i$ at time t to another state $\alpha_{t+1} = S_{i+1}$, depends only on the current state. The sequence of random variables generated by a Markov process is known as Markov chain.

The basic idea of the simulated annealing algorithm is to use random search in terms of a Markov chain, which not only accepts changes that improve the objective function, but also keeps some changes that are not ideal. In a maximization problem, for example, any better moves or changes that decrease the value of the objective function f will be accepted however, some changes that decrease f will also be accepted with a probability P. This probability P, also called the transition probability is determined

$$P = e^{-\frac{\Delta E}{kT}}$$

, where k is the Boltzmann's constant, and let say for simplicity, we can use value of k to be 1. T is the temperature for controlling the annealing process. E is the change of the energy level. This transition probability is based on the Boltzmann distribution in statistical mechanics.

The simplest way to link E with the change of the objective function Δf is to use

$$\Delta E = \gamma \Delta f$$

where γ is a real constant. For simplicity without losing generality, we can use γ = 1. Thus, the probability P simply becomes

$$P = e^{-\frac{\Delta f}{T}}$$

Whether or not we accept a change, we usually use a random number r as a threshold. Thus, if P > r

$$P = e^{-\frac{\Delta f}{T}} > r$$

the move will be accepted.

3.3.1 Application of Simulated Annealing

Simulated Annealing has been successfully applied in multiple domains of engineering like design optimization of automobile suspension system [49], structural optimization, water distribution system, circuit board design problem and selection of fixture elements. Simulated Annealing has also been applied to popular computer problems like maximum cut problem and independent set problem. However, due to single solution tendency of Simulated Annealing with main principle rely on Markov's chain simulated annealing tends to struck at local optima's.

SA has already been implemented in various domain of software engineering like software quality estimation, software reliability forecasting, software requirements prioritization and software project time line design. Simulated Annealing algorithm principle is used more as an algorithm enhancing tools rather than self-independent algorithm because of its single population nature.SA is also implemented to solve test case prioritization problem since nature of the problem is very similar to travelling salesman problem and TSP can be very easily solved by simulated annealing algorithm

3.3.2 Algorithm of SA for Test Prioritization

Formulate a parametric table, with rows defining the test cases and columns defining the faults covered by those test cases for Jmeter's data set.

Objective function f(x): Average Percentage Fault Detection Metric

Terminating Condition: Number of iterations

I. Initialize initial temperature T_0 and initial guess initial starting solution
II. Set final temperature T_f and max number of iterations N
III. Define cooling schedule $T = \alpha T$, $(0 < \alpha < 1)$
IV. While ($T > T_f$ and $n < N$)
    a. Move to new test case ordering: = x + e, where e is selected to be 2-opt or double bridge move based on Roulete Wheel Selection
    b. Calculate $\Delta f = f_{n+1}(x_{n+1}) - f_n,(x_n)$
    c. Accept the new solution if better
    d. if not improved

i. Generate a random number r

ii. Accept if $P = e^{-\frac{\Delta f}{T}}$

e. end if

f. Update the best solution

g. n = n + 1

V. End While

VI. Best solution obtained in all iterations represents the prioritized order of test cases

VII. Identify the redundant test cases using the following function:

a. From least ranked test case to highest ranked test case:

i. If faults covered by test case is already covered by higher ranked test cases
Remove the test case

ii. Else
Add test case to front of test cases to be executed

VIII. Final list contained the prioritized and minimized set of test cases

3.4 Cuckoo Search

Cuckoo search (CS) is among the latest nature-inspired meta-heuristic algorithms, based on the brood parasitism of some cuckoo species. In addition, this algorithm is further enhanced by the Lévy flights rather than using simple isotropic random walks for movement. Recent studies show that CS is potentially far more efficient than its pear of the same group like PSO and genetic algorithms. For optimization problem, the quality or fitness of a solution can simply be proportional to the value of the objective function. Forms of fitness can be defined in a similar way to the fitness function defined in genetic algorithms. The idea of Cuckoo Search algorithm is to mimic this behavior of cuckoos with virtual simulated agents walking from generation over generation, representing the problem to be solved.

Cuckoos are fascinating birds not because of the sounds they make but also because of their aggressive and unique reproduction technique. Few cuckoo species like ani and Guira lay their eggs in communal nests i.e. nest managed by some other bird, though they may eliminate other's eggs to increase their own egg hatching probability [50].

There are three types of brood parasitism: nest takeover, co-operative breeding and intraspecific brood parasitism. A notable number of species engage in obligate brood parasitism by laying their own eggs in the nests of other host birds more often in some other species nest. Some host birds can engage in direct conflict with the intruding cuckoos. If a host bird found that the eggs are not their own, they will either throw away found alien eggs or simply leave its own nest and build a new nest somewhere else. Some cuckoo species such as the New World brood-parasitic Tapera have evolved to such extent that they can mimic colour and pattern of the eggs of a few chosen host species. This reduces the probability of their eggs being destroyed and thus increases their reproduction probability. Timing of laying egg of some species is also fascinating. In general, the cuckoo eggs hatch slightly earlier than their host eggs. Once the first cuckoo offspring is hatched, the first action it will take is to evict the host eggs by blindly throwing the eggs out of the nest, which increases the cuckoo offspring's share of food provided by its host bird. Cuckoo search idealized such breeding behaviour, and thus can be applied for various optimization problems.

Cuckoo Search is simulated by set of independent nests. Eggs in the nest represent an independent separate solution and a cuckoo egg represents a possible new solution. The goal is to use the new potentially better solutions (cuckoos) to replace a bad solution in the nests. For simplicity, it can be considered that each nest contain one egg that is, one solution but in the case of optimizing multi-objective function multiple egg in a nest represents multiple solution corresponding to several objectives. The algorithm can further be extended to more complicated cases according to the need of situation and problem.

CS is idealized based on these three simple rules:

- Each cuckoo generates one solution at a time, and dumps it into a randomly chosen nest
- The nests containing the best solutions possessing high quality will be carry over to the next generation only
- Total number of available hosts nests are fixed, and the solution generated by a cuckoo is discovered by the host bird with a probability P € [0, 1]. For simplicity this assumption can be approximated by the fraction P of the total n nests must be replaced by new nests with random new solution

Studies have shown that flight behavior of many animals, insects and bird possessed the typical characteristics of Levy flights demonstrating a straight flight paths punctuated by a sudden 90^o turn. Levy flights are random walks step length of whose is driven by Levy distributions. When such behavior was applied to optimization and optimal search algorithms then results show promising capability. Similar is the case with Cuckoo Search, when Levy Flight is incorporated with CS performance has been improved drastically. Levy Flights maximizes the efficiency of searches in unpredictable environments.

New solutions i.e. cuckoo are generated via levy flights using following equation

$$x(t+1) = x(t) + \alpha * Levy$$

where $\alpha > 0$ is the step size which should be scaled to the problem of interest.

3.4.1 Application of Cuckoo Search

Cuckoo Search has shown promising efficiency into engineering optimization problems. For example, for problems like spring design, structural optimization and welded beam design cuckoo search achieved better results than existing algorithms. Cuckoo search algorithm is discretised to solve nurse scheduling problem, travelling salesman problem [51], Knapsack problems. Cuckoo search has been proposed for data fusion in wireless sensor networks. Comparison of the cuckoo search with other computational algorithms like Particle swarm optimization, Differential evolution and Artificial bee colony algorithm shows that Cuckoo search provide more robust results than PSO and ABC algorithm. Cuckoo Search is highly suitable for large scale problems. CS is successfully applied to train neural networks with improved performance. More recently, cuckoo search algorithm is used for solving boundary value problem.

Cuckoo search was illustrated to efficiently generate independent test paths for structural software testing [52] and test data generation [53]. In addition, a new software testing approach has also been proposed based on cuckoo search. Like PSO, GA and ACO Cuckoo Search has already been implemented in various domain of software engineering like software quality estimation, software requirements prioritization, software project time line design and test suite prioritization as it can be used to solve problems in less time with much lesser complexity and hassle. Since nature inspired algorithm are efficient in time constraint optimization problems, CS

performed excellently in prioritization of test cases. Below, we have provided an algorithm of CS for solving Test Suite Optimization (TSO) problem. In solving TSO, the word nest is synonymous to one test case ordering.

3.4.2 Algorithm of CS for Test Prioritization

We used the simple representations that each egg in a nest represents a potential test case ordering, and a cuckoo egg represents a new test case ordering with the aim is to use the new and potentially better solutions to replace bad solutions in the nests. Levy Flight is implemented by making a double-bridge move.

Fitness function f(x): Average Percentage Fault Detection Metric

Terminating Condition: Number of iterations

I. Generate initial population of n host nests each nest representing one test case ordering
II. while (Number of iterations)
    a. Get a cuckoo (new solution) randomly by Levy flights. Levy here represents double bridge move and 2-opt move
    b. Evaluate its fitness F_i
    c. Choose a nest among n (say, j) randomly
        i. if ($F_i > F_j$)
            Replace j by the new solution;
        ii. End if
    d. Fraction P of worse nests are abandoned and new ones are built;
    e. Keep the best solutions (or nests with high APFD value);
    f. Rank the solutions and find the current best
III. End While
IV. Best solution obtained in all iterations represents the prioritized order of test cases
V. Identify the redundant test cases using the following function:
    a. From least ranked test case to highest ranked test case:
        i. If faults covered by test case is already covered by higher ranked test cases
            Remove the test case
        ii. Else
            Add test case to front of test cases to be executed
VI. Final list contained the prioritized and minimized set of test cases

Chapter 4

Grey Wolf Optimizer (GWO)

In this chapter we will elaborate the Grey Wolf Optimization technique, its applications in engineering and its general algorithm to solve optimization problems.

Grey Wolf Optimizer (GWO) is a meta-heuristic inspired by the hunting and leadership system of grey wolves, particularly Canis lupus belonging to Canidae family. Hunting mechanism of wolf is successfully modeled mathematically to solve optimization problems as Grey wolves are thought of as pioneer predators leading the food chain at the top alongside tigers and lions.

4.1 Inspiration for Grey Wolf Optimizer

Grey wolves live in a pack and depend on each other for food because they hunt in group. Grey wolves are categorized into four types which are: alpha, beta, delta, and omega based on their leadership hierarchy and social status. The group size is generally 5–12 on average.

4.1.1 Social Hierarchy in Grey Wolves

The ones, male or female, leading the pack are called alphas. An Alpha is responsible for taking trivial decisions like when and what to hunt, where to sleep, at what time to wake i.e. all trivial decision necessary for survival. Decision taken by alpha is followed by the complete pack. The alpha wolves are called the dominant wolf since their instructions has been followed by the pack. But surprisingly, alphas may not be the extra ordinary member of the pack but they are suitable for managing the pack hence empathizing that the organization of wolves and discipline in a pack is more importance than its strength.

The second in the hierarchy comes are beta wolves. Betas are helping wolves to the alphas in taking decisions and other activities of the pack. The beta wolves are stated as next best candidate to become alpha in case alpha wolf retired. The beta wolf respect alpha wolves and take order from alpha, but dictate decisions on lower-level wolves. So they play the role of an advisor to the alpha and gives feedback to the alpha and act as discipline implementer for the group.

The least prioritized grey wolf is omega. Omega wolves take order from all the other more aggressive wolves. Omega may not seem to be important from the perspective of pack but it is observed that pack may become unstable in case of losing the omega. Presence of omegas assists in satisfying the entire pack and maintaining their dominance structure.

If a wolf is none of the above classified wolves then they belong to subordinate group named delta. Delta wolves follow the alphas and betas, but they over power the other left wolves i.e. omega. They are also responsible for watching the privacy and security of group and warning the group when there is possible danger. Grey wolf's shows a very strict social dominant hierarchy as shown in Figure 2

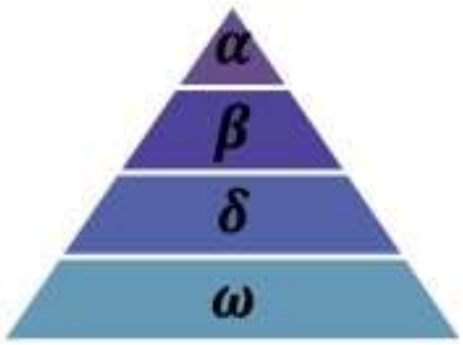


Figure 2 Hierarchy of Grey Wolf [11]

4.1.2 Activities in Grey Wolves

In addition to the social hierarchy of wolves, the other three main steps of hunting viz.

- Looking out for prey,
- Encircling an blocking the prey,
- Attack on prey, are implemented

Figure 3 shows all the above steps

[A] Looking out for prey, chasing, approaching, and tracking prey [B–D] and encircling [E] Attack on prey [11]

Figure 3 Hunting Pattern of grey wolves

Group hunting is a fascinating social behavior of grey wolves. The dominant phases of grey wolf hunting are following:

- Drawing closer to prey.
- Chasing, surrounding, and exhausting the prey until it quit moving.
- Attacking the prey

4.2 Mathematical Mapping of GWO

Mathematical models of the social behavior, tracking, encircling, and attacking on prey by grey wolves are provided as followed.

4.2.1 Social hierarchy

Social hierarchy in the wolves while designing GWO can be outlined as follow:

- Consider alpha (α) as the best solution.
- Similarly, second and third best solutions are chosen as beta (β) and delta (Δ) respectively.

- All reminder solutions in the group are considered as omega (Ω).

In the GWO algorithm the optimization of problem is guided by α, β, and Δ wolves. The reminder Ω wolves tend to move toward these three dominant wolves.

4.2.2 Prey Encircling

As stated already, while chasing grey wolves encircle prey. To mathematically represent encircling mechanism of grey wolf the following equations are proposed [11]:

$$D = |\vec{C}.\hat{X}p\,(i) - \vec{A}.\hat{X}\,(i)|$$

$$X(i+1) = Xp(i) - \vec{A}.\vec{D}$$

Where,

i represents the current iteration,

$\vec{A}$ and $\vec{C}$ are coefficient vectors,

$\hat{X}p$ States the position vector of the prey,

$\hat{X}$ States the position vector of a grey wolf.

The vectors $\vec{A}$ and $\vec{C}$ are obtained as follows [11]:

$$\vec{A} = 2\vec{a}\,.\overrightarrow{r1} - \vec{a}$$

$$\vec{C} = 2\,.\overrightarrow{r2}$$

where value of $\vec{a}$ is linearly decreased from 2 to 0 over each iterations and r1, r2 are random vectors such that r1,r2 € [0, 1].

To illustrate the impact of above two equations, a two-dimensional position vector along with some of the possible neighbors are illustrated in Fig. 4. It can be observed from the figure, a grey wolf standing at position (U, V) can change its position according to the position of the prey which is at (U*, V*). Neighboring place near the current agent can be obtained by adjusting the value of $\vec{A}$ and $\vec{C}$ vectors. For instance, (U*–U, V*) can be obtained by making $\vec{A}$ = (1, 0) and $\vec{C}$ = (1, 1) [11]. The possible updated positions, in three dimensional spaces, of a grey wolf are illustrated in Fig. 3. Random Vectors r1 and r2 allow wolves to obtain any nearby position

between the points shown in Fig. 4. So a grey wolf can update its position inside the space around the prey to any random location by using $D = \left| \vec{C}.\hat{Xp}\,(t) - \vec{A}.\hat{X}\,(t) \right|$ and $X(t+1) = Xp(t) - \vec{A}.\vec{D}$

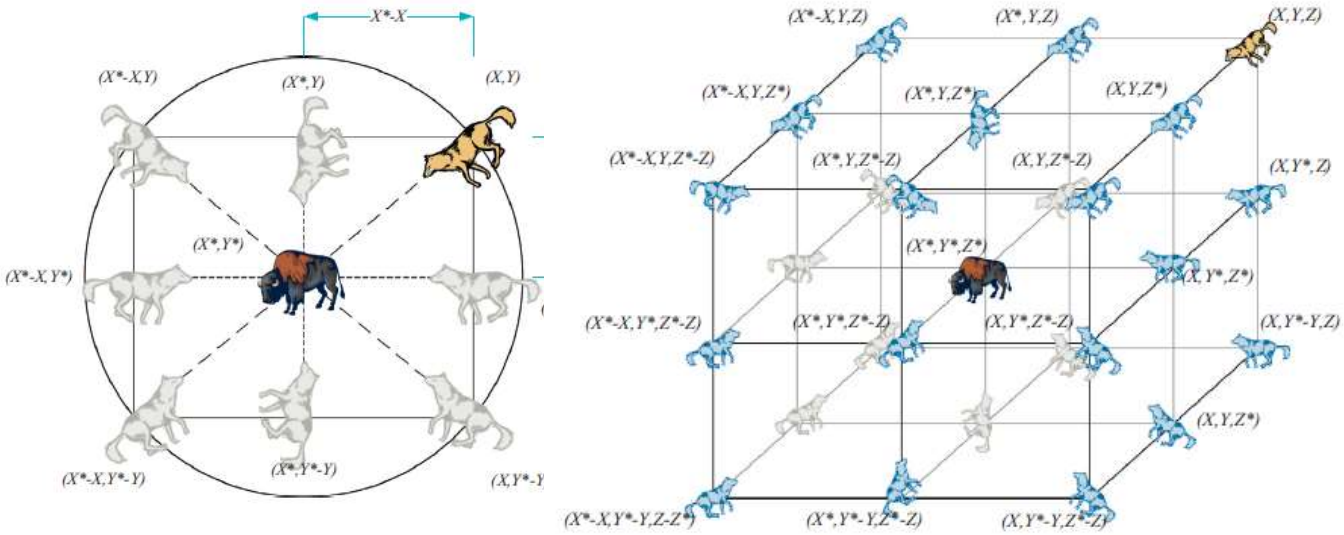


Figure 4 2D and 3D position vectors and their possible next locations.[11]

The n dimensions problems can be solved using similar concept then the grey wolves will move in hyper-cubes because n dimension can be represented in hypercube.

4.2.3 Hunting Mechanism of Grey Wolf

Grey wolves can identify the position of prey and trap them. Alpha made the decision for hunting. The beta and delta may also make hunting decisions sometimes. But practically in unknown search space we don't know the position of the optima's. For mathematically representation of the hunting mechanism of grey wolves, it is considered that the alpha, beta, and delta (best solution among the pack) possess better knowledge about the location of optima. Therefore, we store the first three best solutions obtained so far and other search wolves update their positions according to the position of the alpha agents. The following formulas are proposed in this regard [11]

$$\overrightarrow{D\alpha} = \left| \overrightarrow{C1}\,.\overrightarrow{X\alpha} - \vec{X} \right|$$

$$\overrightarrow{D\beta} = |\overrightarrow{C1}.\overrightarrow{X\beta} - \vec{X}|$$

$$\overrightarrow{D\Delta} = |\overrightarrow{C1}.\overrightarrow{X\Delta} - \vec{X}|$$

$$\overrightarrow{Xa} = \overrightarrow{X\alpha} - \overrightarrow{A1}.\overrightarrow{D\alpha}$$

$$\overrightarrow{Xb} = \overrightarrow{X\beta} - \overrightarrow{A2}.\overrightarrow{D\beta}$$

$$\overrightarrow{Xc} = \overrightarrow{X\Delta} - \overrightarrow{A1}.\overrightarrow{D\Delta}$$

$$\overrightarrow{X}(t+1) = \frac{\left(\overrightarrow{Xa} + \overrightarrow{Xb} + \overrightarrow{Xc}\right)}{3}$$

Fig. 5 illustrates how omega wolf changes its position according to position of alpha, beta, and delta in a two dimensional space. From the figure it is observed that the final position of omega agents would be in a random place in between the positions of alpha, beta, and delta in the search space. In simple, we can conclude that alpha, beta, and delta drives towards the location of the prey.

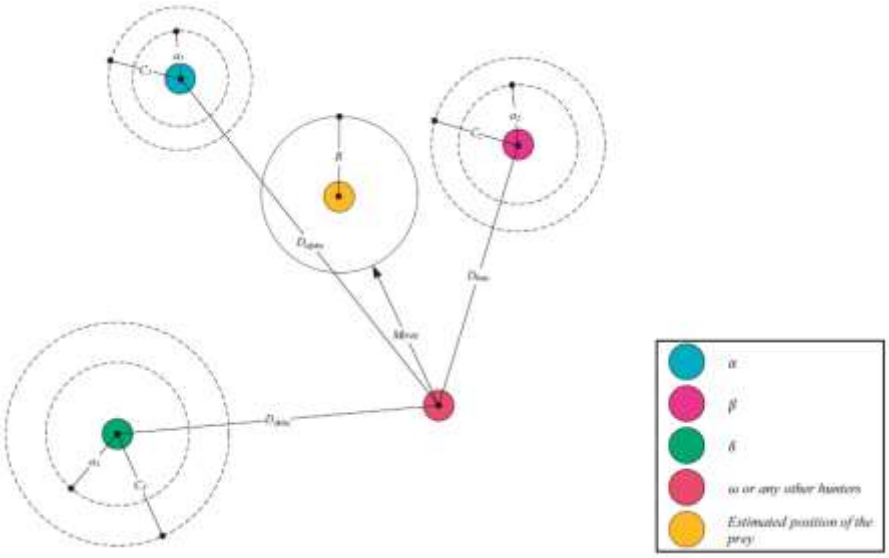


Figure 5. Position updating in GWO [11]

4.2.4 Attacking prey (exploitation)

As stated above the grey wolves complete the hunt process by attacking the prey when it becomes stable. To mathematically represent this, the value of $\vec{a}$ is decreased gradually. Also

value of $\vec{A}$ is in the range [2a, 2a] where value of $\vec{a}$ is changed from 2 to 0 over each new iteration.

Values of $\vec{A}$ as [1, 1] signifies that the next position of a searching agent is towards prey and can be anywhere in between its current position and the estimated position of the prey.

However, the GWO algorithm with the operators proposed so far is tend to stuck at local solutions. It is observed that the trapping mechanism provides exploration, but GWO needs few more for thorough exploration.

4.2.5 Search for prey (exploration)

Grey wolves explore the search space according to the location of the alpha, beta, and delta wolves. They tend to deviate from each other to look out for prey and may be converge to attack the prey. Divergence of the system can be mathematically modeled by making value of $\vec{A}$ greater than 1 or less than -1. This makes exploration strong and allows the GWO algorithm to search globally. Fig. 5(b) also shows that |A| > 1 forces the grey wolves to deviate in hope to find a better prey.

$\vec{C}$ provides random weights for prey and favors exploration and help GWO in enforcing a more random behavior in optimization process favoring exploration and ignoring local optima. An obstacle in the hunting paths of wolves makes approaching few preys difficult. $\vec{C}$ tends to replicate this behavior. Depending on the location of a wolf, it can give the prey a random weight and make it difficult to reach and so is the vice versa.

Following key points summarizes the complete Grey Wolf Optimization process:

- Alpha, beta, and delta wolves locate the probable position of the prey (global solution).
- Every other wolf which can be a solution updates its distance from the prey.
- The value of parameter a is changed from 2 to 0 in to achieve exploration and exploitation.
- Possible solutions diverge from the prey when |A|> 1 and converge towards the prey when |A| < 1.
- Finally, the GWO algorithm is terminated by stopping criteria.

Some points may be noted to illustrate how GWO theoretically solve optimization problems:

- Social hierarchy assists GWO to look out for the best solutions obtained so far over the period of iterations.
- Encircling mechanism defines a probable surrounding around the solution which can be extended to higher 'n' dimensions as a hypercube.
- Hunting method allows possible solutions to identify the position of the prey.
- Exploration and exploitation are guaranteed by the making wise decision for the values of $\vec{a}$ and A.

4.3 Algorithm for Continuous grey wolf optimization algorithm [11]

Input: n Number of grey wolves in the pack,

Stopping Criteria: Number of iterations for optimization

Output: x_α which represents optimal grey wolf position

I. Initialize a population of n grey wolves' positions randomly.
II. Find the α, β and δ solutions based on their fitness values.
III. While Stopping criteria not met do
    a. For each Wolf in pack
    b. do

Update current wolf's position according to $\overrightarrow{X}(t+1) = \frac{(\overrightarrow{Xa} + \overrightarrow{Xb} + \overrightarrow{Xc})}{3}$

    c. end
    d. Update a, A, and C:
    e. Evaluate the positions of individual wolves
    f. Update α; β; and δ:
IV. End While

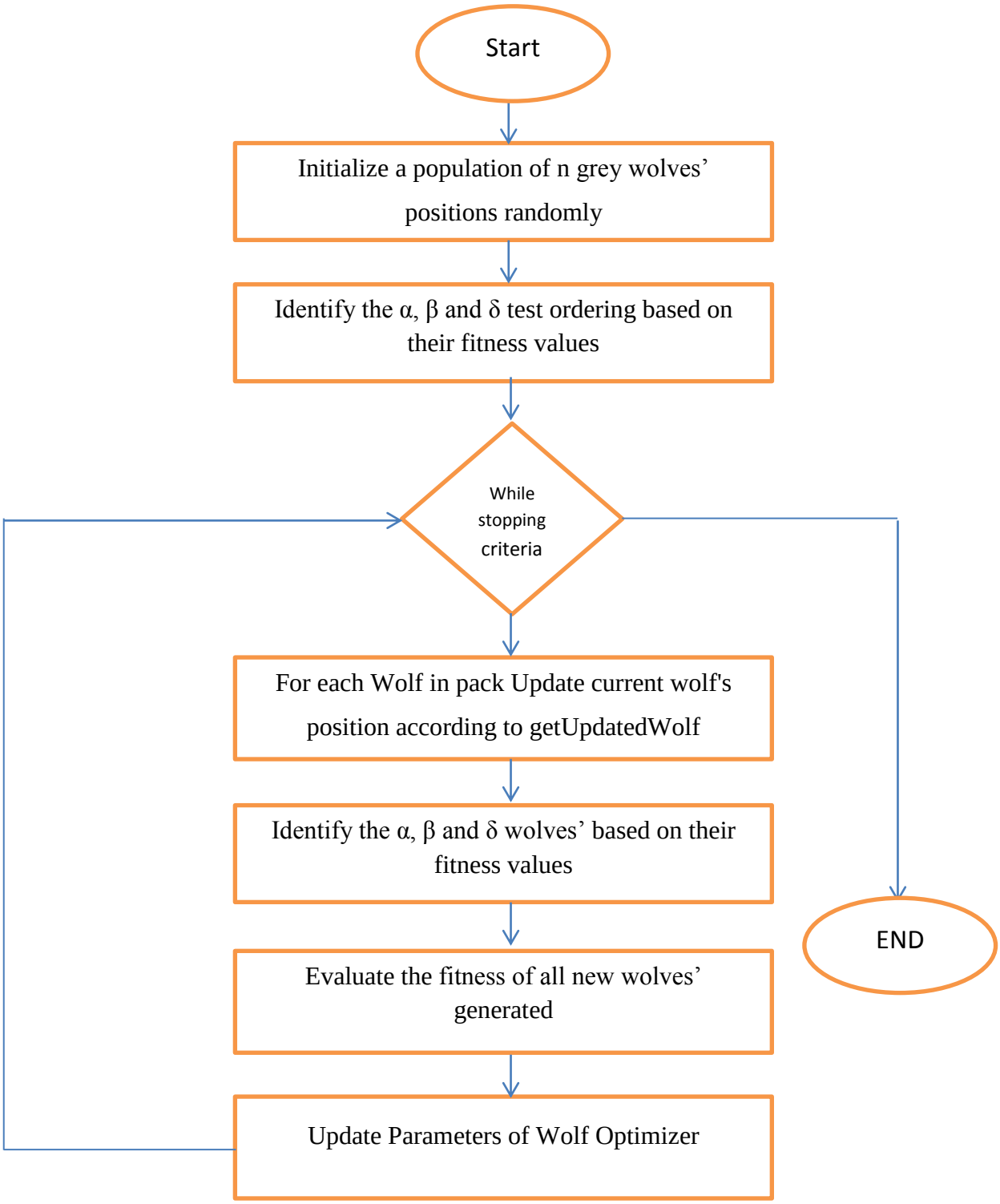


Figure 5 Flow Chart of GWO

4.4 Applications of GWO

GWO has been utilized for various domain of engineering liker training neural nets, for solving economic dispatch problems, feature Subset Selection approach [54]. Vehicle Routing Problem has been solved using GWO. GWO was able to provide highly competitive results compared to well-known heuristics such as PSO, GSA, DE, EP, and ES. Results on real problems also show that GWO shows high performance in both unconstrained problems and constrained problems as well.

Chapter 5

PROPOSED APPROACH TO TEST CASE PRIORITIZATION

This chapter provides the implementation details of the proposed Grey Wolf Optimizer in test prioritization. The detailed explanation pertaining to implementation can be divided into three sections. The first section provides a brief description of how grey wolf is discretized; the second section discusses the algorithm for test prioritization using discretization described in section first.

In this chapter we will cover how Grey Wolf Optimizer is mapped to solve Test prioritization problem. This algorithm is a part of computational intelligence technique and performs optimization which is inspired by the nature.

Test suite prioritization process using Grey Wolf Optimizer is illustrated on real world test suite of Jmeter. Faults covered by these test cases are categorised into 11 classes viz. Resource not found, File cannot be created, Function not found, Invalid variable or Parameter, Invalid Query, Alias, Invalid file access, In valid results, Out of range, End of file and divide by zero. Thus test case formulated to be used as test data for the implemented algorithms. Based on fault detection value and time to implement metric named average percentage of fault detected (APFD) proposed by Elbaum [23] is calculated for each test case ordering to rank the test cases. The ranking thus obtained is used to assist in test case minimization, results of which are explained in next chapters.

5.1 Discrete Grey Wolf

A discretized version of the grey wolf optimization is required to represent the positioning of wolves in respect to ordering of test case. Grey wolf optimizer (GWO) is among the latest nature inspired optimization approaches which mimics the hunting process of grey wolves in nature.

In the continuous grey wolf optimization wolves regularly change their positions to whatever point in the space. The discrete version introduced here is performed using two operators chosen by strategy described below then stochastic crossover strategy is used to identify the updated discrete grey wolf position. This approach for discrete grey wolf optimization is applied in the

test case prioritization domain for finding test case ordering maximizing average Percentage of fault detected metric while minimizing the number of test case selected.

5.1.1 Representation of Test Case Ordering

A solution representation for the test suite prioritization is a permutation representation as illustrated by Figure below.

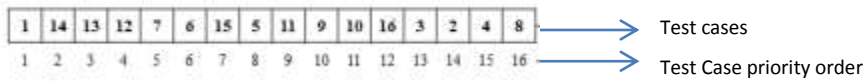


Figure 6 The permutation representation of a solution

In this case, there is no distinction between wolf and a single priority order, as each wolf corresponds to one solution. In this representation, elements of array represents test case and the index represents the order of a tour.

5.1.1.1 The wolf

If we assume that a wolf represents a single solution, we can give wolf the following properties:

- Each wolf is separate independent potential solution represented by one individual in the population.
- In our case wolf represents single test case prioritization order of test cases.
- Difference between the positions of wolves decides the size of steps.
- Every test case can be place after every other test case in all wolves assuming that no test case repeats itself in single prioritization order.

5.1.1.2 The pack

In GWO, the following features can be imposed concerning a pack:

- The numbers of wolves are fixed.
- In our case pack represents a complete set of multiple test case orderings.
- A pack is an individual of the population and the number of wolves in the pack is equal to the size of the population

- Alpha wolf is represented by the fittest solution in the iteration.

By the projection of these features on test prioritization, we can say that a nest is shown as an individual in the population with its own prioritization order.

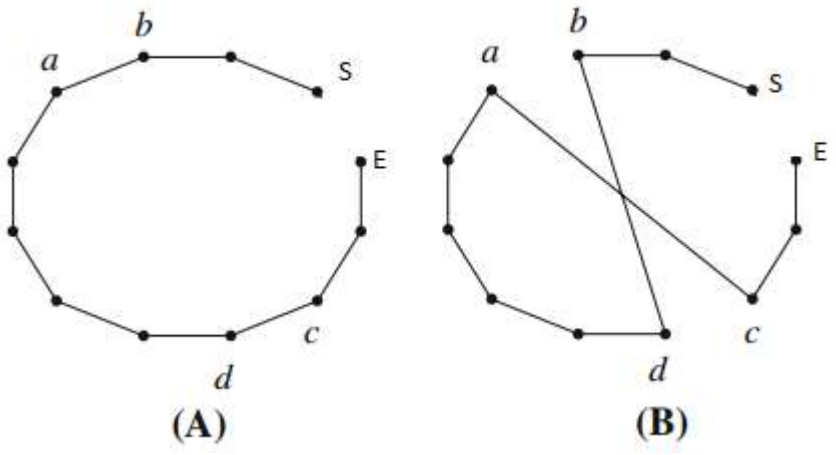
5.1.2 Operators in Grey Wolf

In continuous problems, the meaning of divergence and convergence towards prey is obvious. However, for combinatorial problems, the notion of divergence and convergence requires that the given solution must be generated by the perturbation. Perturbation must make the minimum changes on the candidate solution.

This leads to the 2-opt move and double bridge move for a new solution which are explained as following:

5.1.2.1 2-Opt Operator

The minimum number of non-contiguous edges that we can delete is two. So breaking of two edges and reconnecting them in other way to make new solution with minimum changes as shown in the Figure below. Here dots represent the test cases and ordering of test case is path followed from S to E

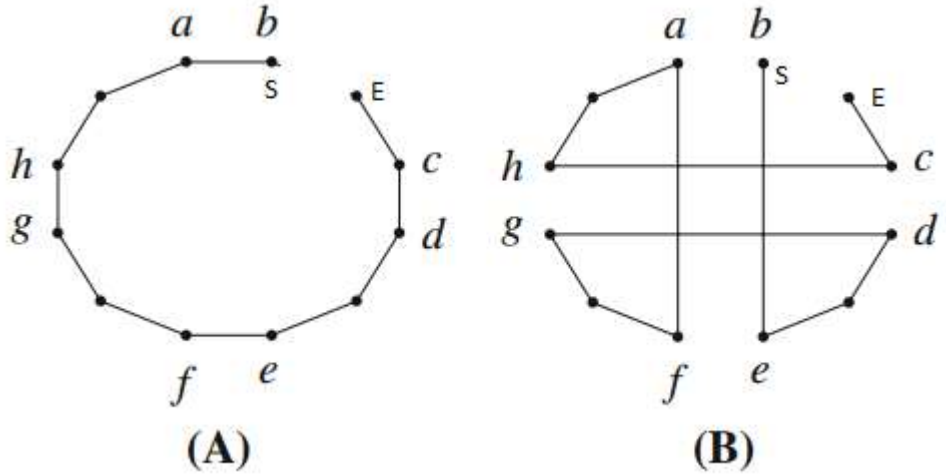


(A) Initial ordering from S to E (B) The ordering created by 2-opt move from S to E [the edges (a, b) and (c, d) are removed, while the edges (a, c) and (b, d) are added]

Figure 7. 2-opt move

5.1.2.2 Double-Bridge Operator

Double Bridge perturbation cuts four edges and introduces four new ones as shown in Figure. Notice that each bridge is a 2-change, but neither of the 2-changes individually keeps the graph connected. Strength of Double Bridge move is four. Here dots represents the test cases and ordering of test case is path followed from S to E



(A) Initial ordering from S to E. (B) The ordering created by double-bridge move [the edges (a, b), (c, d), (e, f) and (g, h) are replaced by the edges (a, f), (c, h), (e, b) and (g, d), respectively]

Figure 8. Double-bridge move.

5.1.2.3 Difference function between two wolves

Distance between the position of two wolves decides that weather a 2-opt move or double bridge move is selected based on the value of variable cstep calculated by following equation

$$cstep = \frac{1}{1 + e^{(-10*(A*D\_wolf - 0.5))}}$$

D_wolf represents the difference in position of two wolves which is calculated by element wise subtraction of two test ordering and then counting number of non-zero elements in the result.

5.2 Objective Function

Each wolf representing a solution in the search space is associated with a numeric objective value. So the quality of a solution is proportional to the value of the objective function. In GWO, a wolf of better quality will attract other wolves to move towards itself. This means that the quality of a wolf is directly related to its ability to converge towards global solution. For the test case prioritization problem, the quality of a solution is related to the value of Average Percentage of fault detected metric. The best solution is the one with the highest average percentage of fault detection value.

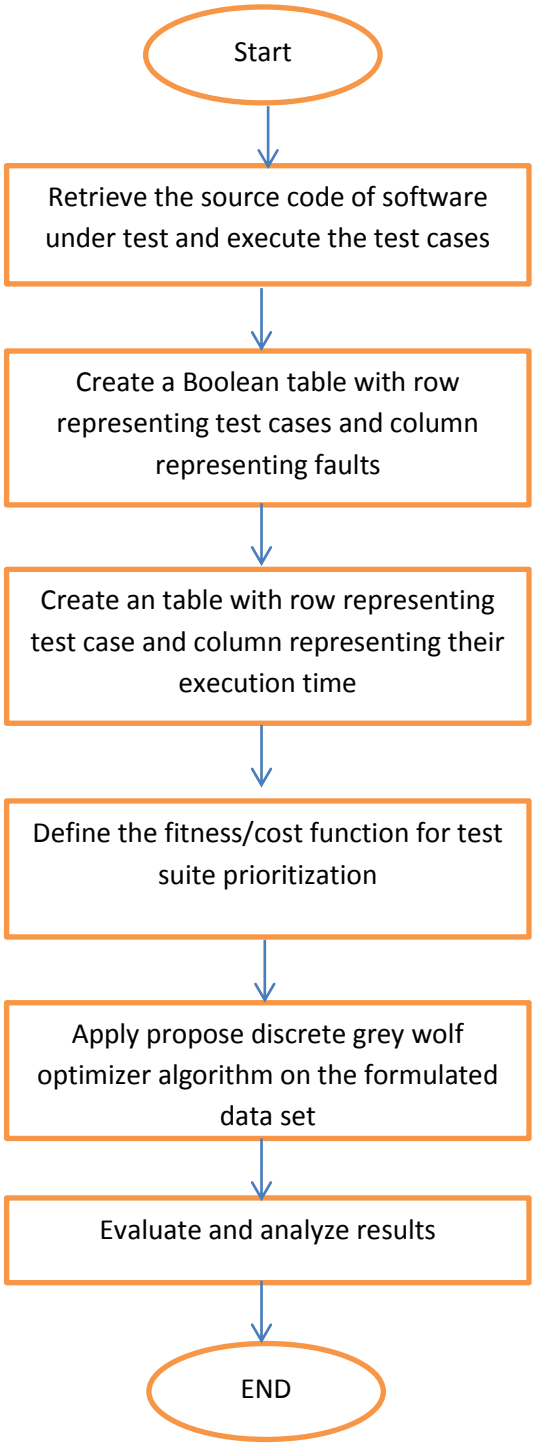
Mathematically,

Let T be a test suite containing n test cases and let F be a set of m faults revealed by T. Let TF_i be the first test case in ordering T' of T which reveals fault i. The Average Percentage of fault detected metric (APFD) for test suite T' is given by the equation [24]:

$$APFD = 1 - \frac{TF1 + TF2 + \quad + TFm}{n * m} + \frac{1}{2 * n}$$

5.3 Framework for Algorithm

This section provides the steps necessary to apply discrete GWO for test suite prioritization. The data set for test suite optimization consists of a binary matrix with rows defining the test cases and columns defining the faults covered by those test cases for Jmeter's data and a separate table represents test cases along with their execution time.



- Create a database file containing the data set on which testing has been performed. It is essential.
- Create a file consisting of m x n matrix where m represents the test cases and n represents faults which are covered by those test cases.
- Define the cost function used in GWO algorithm. In test suite prioritization the cost function is the average percentage of fault detected.
- Grey Wolf optimizer is an open problem optimization algorithm with its generic set of parameter viz. no of wolfs in population which need to be adjusted as per the problem domain.
- GWO population parameter is adjusted as per size of test data
- Cost Function: Fault Detection Rate, that is Number of Faults covered per Execution Time.
- Generation Limit: It defines the number of iteration of the BBO algorithm.
- Elite Solution List: It is multidimensional real-valued matrix which selects best candidate solution that will be preserved for next generation.
- Elite Cost List: the cost associated with the elite solution will be stored in minimum cost list.
- Sorting: The sorting order should be either descending or ascending depending on the problem. In our case we need to maximize APFD metric so descending order is desired.
- Apply the Discrete Grey Wolf Optimizer algorithm as stated above
- Evaluate the performance by identifying the minimized set of test cases and priorities that can be assigned to the test cases.
- The algorithm can be applied to larger data set.

5.4 Algorithm of Proposed GWO for Test Prioritization

Formulate a parametric table, with rows defining the test cases and columns defining the faults covered by those test cases for Jmeter's data set.

Objective function f(x): Average Percentage Fault Detection Metric

Input: n Number of grey wolves in the pack,

Stopping Criteria: Number of iterations for optimization which depends upon the size of optimization data

Output: x_α which represents optimal grey wolf test case ordering

I. Initialize a population of n grey wolves' positions randomly. Each wolf representing a test case ordering such that value in ordering € [1 no_testcases]
II. Find the α, β and δ solutions based on their fitness values.
III. While Stopping criteria viz. number of iterations for optimization are not reached do
    a. For each Wolf in pack
    b. do
        Update current wolf's position according to getUpdatedWolf procedure
    c. end
    d. Update a, A, and C
    e. Evaluate the positions of individual wolves
    f. Update α; β; and δ:
IV. End While
V. Identify the redundant test cases using the following function:
    a. From least ranked test case to highest ranked test case:
        i. If faults covered by test case is already covered by higher ranked test cases
            Remove the test case
        ii. Else
            Add test case to front of test cases to be executed
VI. Final list contained the prioritized and minimized set of test cases

5.4.1 Algorithm for getUpdatedWolf Procedure

The main updating equation can be formulated as stated below:

$$cstep = \frac{1}{1 + e^{(-10*(A*D\_wolf - 0.5))}}$$

Where A is calculated used equation

$$\vec{A} = 2\vec{a} \,.\, \overrightarrow{r1} - \vec{a}$$

and D_wolf is calculated using difference in position of two wolves which is calculated by element wise subtraction of two test ordering and then counting number of non-zero elements in the result.

Algorithm:

I. Define two random numbers r1 and r2 € [0, 1]

II. Calculate the value of A1,A2,A3 and C1, C2, C3 using equation $\vec{A} = 2\vec{a} \cdot \overrightarrow{r1} - \vec{a}$ and $\vec{C} = 2 \cdot \overrightarrow{r2}$ for different values of r1 and r2 while calculating

III. Calculate cstep(α), cstep(β), cstep(δ) using

$$\text{cstep(x)} = \frac{1}{1 + e^{(-10*(A*D\_wolf(x)-0.5))}}$$

IV. For alpha, beta, delta

a. If cstep>=rand

Choose Double Bridge move

b. Else

Choose 2-opt move

V. End for

VI. Perform Crossover(x, y, z) using simple stochastic crossover strategy

$$X=\begin{cases} x \; if \; rand < \frac{1}{3} \\ y \; if \; \frac{1}{3} < rand < \frac{2}{3} \\ z \; if \; rand > \frac{2}{3} \end{cases}$$

VII. Return X as updated wolf position

5.5 Flow Chart of Discrete Grey Wolf Algorithm

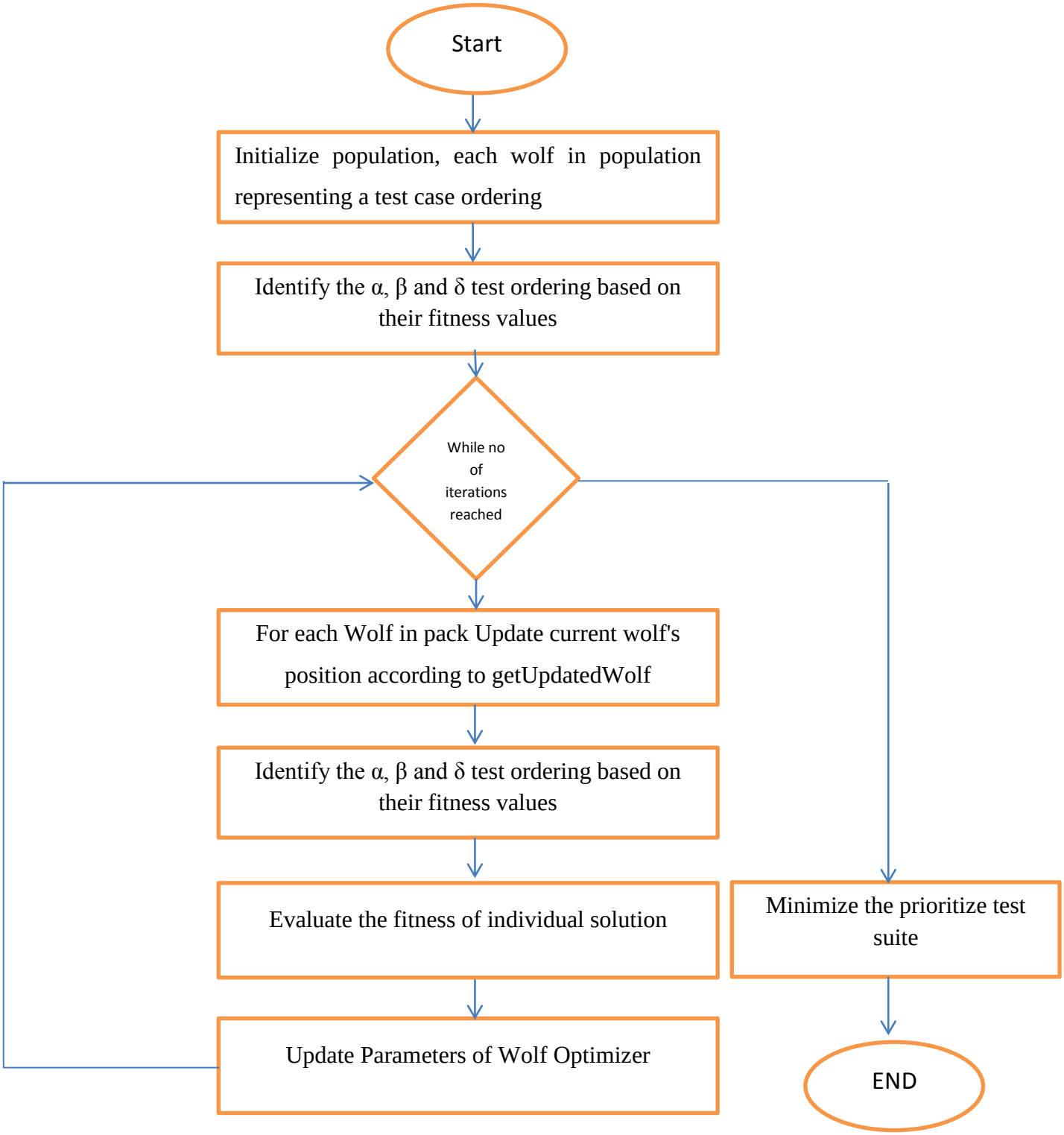


Figure 9. Flow Chart of Discrete Grey Wolf Algorithm

5.6 Discussion on Proposed Algorithm

1) Generate test case population. Formulate a parametric table, with rows defining the test cases and columns defining the faults covered by those test cases.

For example, consider there are 10 test cases A, namely A, B, C, D, E, F, G, H, I, J with following fault identification table

TC/Fault	F1	F2	F3	F4	F5	F6	F7	F8
A	0	0	0	0	0	0	0	0
B	1	1	0	0	0	0	0	0
C	1	1	1	0	0	0	0	0
D	0	1	1	0	0	0	0	0
E	1	0	0	0	0	0	0	1
F	0	0	0	0	0	0	0	1
G	0	1	0	0	0	0	0	0
H	0	0	0	1	0	0	0	1
I	1	1	1	1	1	0	0	0
J	0	0	0	0	1	1	1	0

2) Initialize a random pack of 'n' wolves
    a. Wolf represents single test case prioritization order of test cases
    b. Pack represents a complete set of multiple test case orderings.
    c. The number of wolves in the pack is equal to the size of the population
    d. Alpha wolf is represented by the fittest solution in the iteration.
    e. Beta and Omega are represented by second and third best in the pack
    f. Stopping Criteria is no of iteration done to update wolf ordering

Let there are four wolves in population with test case ordering represented as follow:

$$Pack \quad \begin{cases} \text{Wolf 1: A} - \text{B} - \text{G} - \text{H} - \text{I} - \text{J} - \text{C} - \text{D} - \text{E} - \text{F} \ (62.5)\ omega \\ \text{Wolf 2: I} - \text{J} - \text{E} - \text{B} - \text{C} - \text{H} - \text{A} - \text{D} - \text{F} - \text{G} \ (85)\ \ alpha \\ Wolf\ 3\text{: A} - \text{C} - \text{D} - \text{E} - \text{J} - \text{I} - \text{B} - \text{F} - \text{G} - \text{H}\ (66.25)\ beta \\ Wolf\ 4\text{: C} - \text{B} - \text{E} - \text{A} - \text{D} - \text{F} - \text{G} - \text{H} - \text{J} - \text{I}\ (53.7)\ delta \end{cases}$$

Wolf 2 with high average percentage of fault detection value will become alpha and act as the goal position for all the rest of wolves in the population and Wolf 1,3,4 then try to become closer

to Wolf 2 by updating itself according to the ordering of alpha. Wolf 2 also update itself for exploitation. Fitness Values are calculated using equation $APFD = 1 - \frac{TF1+TF2+\quad +TFm}{n*m} + \frac{1}{2*n}$

For Wolf 1 fitness value can be calculated as follows:

$$APFD = 1 - \frac{2+2+5+4+5+6+6+9}{10*8} + \frac{1}{2*10} = 0.625\ i.e\ 62.5\%$$

Similarly, fitness value of all wolf can be calculated..

3) Implementation

   a. Process repeats till any stopping criteria satisfy. For the considered problems stopping criteria are the maximum number of iterations. In the proposed algorithm maximum numbers of iterations for processing the algorithm are fixed before processing the algorithm.
   b. Update every wolf according to 2-opt or double bridge move
   c. Calculate fitness value of test case sequence using following

$$APFD = 1 - \frac{TF1 + TF2 + \quad + TFm}{n * m} + \frac{1}{2 * n}$$

   d. Rank all the solution and find the optimally best result.

Algorithm converges after two iterations which is stopping criteria in our case. In current iteration every wolf changes its ordering including alpha to explore local search space using getUpdatedWolf Procedure. Now let say position of wolf becomes using 2-opt operator as follows where in Wolf 1 where edge between B-G and J-C are removed in Wolf 1 and replaced by edge B-C and F-G respectively. Similarly C-H and A-D are replaced by C-D and G-H respectively and recalculate the fitness value as calculated before. Marking in red color below shows the change in test case ordering for all wolves.

Wolf 1: A–B– C–D–E–F–G–H–I–J (43.7%) *delta*

Wolf 2: I–J–E–B–C–D–F–G–H–A (90.0%) *alpha*

Wolf 3: A-C-B-F-G-D-E-J-I-H (51.25%) *omega*

Wolf 4: I-A-D-F-J-E-B-C-H-G (81.25%) *beta*

Now Wolf 2 will become alpha wolf in next iteration because of high fitness value obtained and thus represent the location of goal point for ordering updation.

If we set no of iteration for algorithm to two then the above wolf 2 represents the suggested prioritized optimal order of test cases.

So prioritize order of test case is given by I–J–E–B–C–D–F–G–H–A

4) Minimize the prioritize order of test cases.

Reduced set of test case are selected using the reduce function which is based on whether the faults covered by current test are covered by higher prioritize test case or not if faults are covered by higher prioritized test case then current test case is excluded using the following function

$$\sum_{j=1}^{i-1} Tj \oplus Ti = 1 \;\; then\, Ti\, is\, not\, selected$$

Where Ti is the test case to be selected and Tj are the test case having greater priority to the test case Ti. ⊕ operator here identifies that whether test case all faults covered by Ti are covered by Tj or not, if covered then this function return 1 and test case Ti is excluded. Above stated function is applied on prioritized order I–J–E–B–C–D–F–G–H–A. So faults covered by test cases B,C,D,F,G,H,A are already covered by higher prioritized test case I,J,E so they can be excluded and hence test suite is minimized the reduced test case can be given as follows:

Reduced Test Case: I-J-E

In the following chapter the results obtained after applying the proposed algorithm, simulation environment require for algorithm execution are discussed.

Chapter 6

Dataset, Simulation Environment, Results and Discussion

This chapter also provides a summarization of results obtained after applying GWO on test data and thus compares it to the previously applied meta-heuristic techniques. The implementation details of GWO in test prioritization are already discussed in previous chapter.

The proposed discrete versions is compared to two of the common optimizers used in this domain namely Ant Colony Optimization (ACO), Cuckoo Search (CS), Biogeography based optimization (BBO) which falls under population based meta-heuristics and Simulated Annealing (SA) which is trajectory based single population optimization problem. A set of assessment indicators are used to evaluate and compared the different techniques over Results prove the capability of the proposed discrete version of grey wolf optimization to generate test prioritization order regardless of the initialization and the used stochastic operators

6.1 Introduction to JMeter

Apache JMeter is a completely Java based computer application was initially designed for load testing functional behavior and measure their performance.

JMeter is a load testing tool developed by Apache project for analyzing the performance of different type of services with emphasis on applications developed for web since it was originally designed for web and is later extended to other domains like JDBC database connections, FTP, JMS,.HTTP etc.

6.1.1 Dataset based on Jmeter

Dataset is formed by compiling the Jmeter's source code and thus executing Junit's unit test cases to formulate each data point where each row represents the test case and each column represents the faults covered by those test cases. A Boolean matrix is prepared where 1 signifies that the fault is covered by the test case.

Faults covered by these test cases are categorised into 11 classes viz. Resource not found, File cannot be created, Function not found, Invalid variable or Parameter, Invalid Query, Alias,

Invalid file access, In valid results, Out of range, End of file and divide by zero. Thus test case formulated to be used as test data for the implemented algorithms. Based on fault detection value and time to implement metric named average percentage of fault detected (APFD) proposed by Elbaum [21] is calculated for each test case ordering to rank the test cases. The ranking thus obtained is used to assist in test case minimization.

TC/Fault	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	Test Case	Time (ms)
TC1	1	1	1	0	0	0	0	0	0	0	0	TC1	1
TC2	1	0	0	0	0	0	0	0	0	0	0	TC2	6
TC3	0	0	0	1	0	0	0	0	0	0	0	TC3	1
TC4	1	0	0	1	1	0	0	0	0	0	0	TC4	1
TC5	1	0	0	0	0	1	0	0	0	0	0	TC5	4
TC6	0	0	0	0	0	0	1	0	0	1	0	TC6	1
TC7	1	0	0	0	1	0	0	0	0	0	0	TC7	2
TC8	0	0	0	0	0	1	0	1	0	0	0	TC8	1
TC9	0	0	0	0	0	0	0	0	1	0	0	TC9	1
TC10	0	0	0	1	0	0	0	1	0	0	0	TC10	1
TC11	1	0	0	0	0	1	0	1	0	0	0	TC11	2
TC12	0	0	0	0	0	0	0	0	0	1	0	TC12	1
TC13	0	0	0	0	0	1	0	1	0	0	0	TC13	1
TC14	1	0	0	0	0	0	0	0	0	0	0	TC14	1
TC15	0	0	0	0	0	0	0	1	0	0	0	TC15	1
TC16	0	0	0	0	0	0	0	0	1	0	0	TC16	1
TC17	0	0	0	1	0	0	0	0	0	0	0	TC17	1
TC18	0	0	0	0	0	0	0	1	0	0	0	TC18	104
TC19	0	0	0	0	1	1	1	1	0	0	0	TC19	16
TC20	0	0	0	0	1	1	1	1	0	0	0	TC20	6
TC21	0	0	0	1	0	0	0	0	0	0	0	TC21	1
TC22	0	0	0	0	0	0	0	1	0	0	0	TC22	1
TC23	0	0	0	0	0	0	0	0	1	0	1	TC23	2
TC24	0	0	0	1	0	0	0	1	0	0	0	TC24	2

Data set for prioritization

The above figure shows a sample test data which was used to during the research. F stands for Fault Covered and Time stands for Execution Time(ET) of test case. The highest domain value in ET is 104 and lowest domain value is ET is 1.

6.2 Simulation Environment

The section provides information about the tools used in deriving the results in this research. The parameters based on which the comparison was done are Number of initial parameters required, Number of test cases identified, Convergence rate of algorithm, Variance in the final result on multiple runs.

The data set was generated after the application of 24 test cases on 11 faults in Jmeter. Different parameters of meta-heuristic algorithms were taken in to consideration viz. mutation probability, elite solution, generation limit, cost function, emigration probability, and immigration probability, population size and processing time in term of total actual iterations.

6.2.1 Brief Description about MatLab

MATLAB is a high performance language for technical computing. It integrates computation, visualization and programming in an easy-to-use environment where problems and solutions are expresses in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm, simulation and prototyping
- Modeling, simulation and prototyping
- Data analysis, exploration and visualization
- Scientific and engineering and visualization
- Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations in a fraction of the time it would take to write a program in a scalar non interactive language such as C or FORTRAN.

S.No	Simulation Variables	Value
1.	Programming Language	MATLAB
2.	Data Base	Excel
3.	Data Set Repository	Jmeter

Table 1 Simulation Environment for the Research work

6.3 Results and Analysis

The section shows the results obtained. The research was carried out in a system with 2 GB RAM, 500 GB HDD and 1.6 GHz core i5 Intel Processor.

S.no	Parameters	BBO	ACO	GWO	Cuckoo	SA
1.	Mutation Probability	0.4	-	-	-	-
2.	Elite Solution	Best habitat	Phoneme Matrix	Alpha, beta delta	Best cuckoo	
3.	Generation Limit	1000	100	1000	1000	1000 /12
4.	Cost Function	Average Percentage of fault detected	Average Percentage of fault detected	Average Percentage of fault detected	Average Percentage of fault detected	Average Percentage of fault detected
5.	Discovery rate of alien eggs/solutions	0.2 (keep rate)	-	-	0.25	-
6.	Population	10	10	15	15	1
7.	Initial temp	-	-	-	-	0.025
8.	Alpha	-	1	-	-	0.99
9.	Beta	-	1.5	-	-	-
10.	Rho	-	0.05	-	-	-

Table 2 Parameter setting during research

```
[1 20 23 6 21 13 19 24 12 4 5 22 9 11 17 7 3 15 8 10 16 18 2 14]
```

Priority order of test cases

The above figure shows the priorities order of the test case. After simulation of the algorithm on the data set for 30 iterations, we obtained the prioritized order. The priorities are ordered in ascending order with test case at index 1 as the highest priority and 24 being the least. The result showed that test case 1 is much more important than rest of test cases. Test case 14 is least prioritized in testing the concerned application.

```
1     20     23     6     21
```

Reduced Test Cases

Test cases after test case 21 in priority order has been reduced to zero which showed that these test case can be excluded during regression test suite selection. These test cases are found to be redundant and have been identified as worthless by Grey Wolf Optimizer.

After careful evaluation of figure drawn below it is evident that modified GWO has higher tendency towards test suite minimization juxtaposes with the tendency to achieve complete coverage.

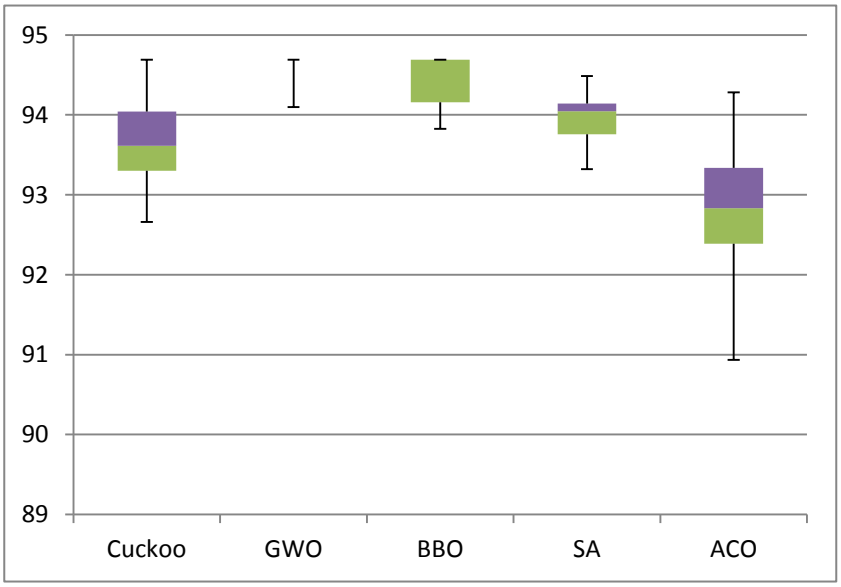


Figure 7 Box Graph of test case ordering for various algorithms (in term of APFD metric)

As evident from the above box graph Grey wolf shows the least variation in the APFD value with most of the results lie near the median resulting into vary sharp 25% and 75% boxes around the median making them even unrecognizable in graph with minimum value of APFD being maximum all the algorithms making GWO most efficient.

	Cuckoo	GWO	BBO	SA	ACO
Maximum	94.68804	94.68804	94.68804	94.48435	94.28066
Minimum	92.66188	94.09841	93.82326	93.3194	90.93589
Variance	0.251797	0.035818	0.098946	0.087572	0.523622
SD	0.501794	0.189257	0.314556	0.295926	0.723617

Table 3 Maximum, Minimum and Variance in APFD value in various algorithms

Moreover from the above table APFD value of GWO is least varied in comparison to other algorithms. ACO shows highest variance in performance making it least stable. However in 30 runs Cuckoo GWO and BBO all obtain similar maximum APFD value. Minimum APFD over 30 runs is highest in GWO making GWO most promising even in its vulnerable state.

S.no	Characteristics		BBO	ACO	GWO	Cuckoo	SA
1.	Complete coverage test cases identified (low is better)		5	5	5	5	5
2.	Number of initial parameters required		4	7	2	3	4
3.	Redundant test case identified (high is better)		19	19	19	19	19
4.	Total Processing Iterations		24000	24000	10000	33125	12000
5.	Total Execution Time of reduced test cases	Worst	25	113	21	113	23
		Best	6	6	6	6	6
6.	Average Convergence rate of algorithm (in term of processing iterations)		19002	12989	3069	18935	11723
7.	Variance in the APFD value		0.098946	0.523622	0.035818	0.251797	0.087572
8.	Reduction in test suite		79%	79%	79%	79%	79%

Table 4 Comparison Table of various algorithms based on different factors

Since all algorithms compared here are efficient meta-heuristic algorithms so number of worthless test case identified and usable test case are same for all algorithms but the major difference is present in convergence rate of the algorithm in term of internal processing requirement which shows that GWO requires least processing with minimum resource requirement.

Moreover the convergence rate of GWO is also high which can also be empirically implied from the above stated table. So, GWO perform considerably better than all algorithms stated above in term of speed also. In fact the combined execution time of reduced test cases even in the worst case is minimum in case of GWO. However in best case the proposed time is same because of meta-heuristic problem solving capability of all algorithms.

Chapter 7

CONCLUSION AND FUTURE WORK

This chapter discusses the conclusions inferred from this research and presents the possibilities of extension of this work in future.

Regression Testing is very crucial activity to bring about success in software development after some minor or major defect has been encountered during execution. It is generally carried out as a maintenance phase activity, which is considered as the most expensive phase, if premeditation about the phase has not been carried out.

There has been large algorithm been proposed and used in the refining the process of regression testing but all these algorithm suffers from constraint problem, elitism problem, loss of coverage and determining the redundant test cases in the test cases. Grey wolf optimization is meta-heuristic technique being implemented and used widely in various field like remote sensing, job scheduling, classification and selection.

This novel technique has been modified in this paper and compared with other already acclaimed meta-heuristic techniques. GWO algorithm produced sufficient reduction in test cases after its execution irrespective of the setting of initial parameters. In generations we obtained, state of the art result. GWO is compared with ACO,BBO,SA and Cuckoo for test case minimization.

GWO can be extended to multi-objective test suite optimization by altering the comparison mechanism using the pareto dominance and optimal front operator where a single wolf contain multiple solution and nature of wolves are identified using these pareto operators.

REFERENCES

[1] Hla, Khin Haymar Saw, YoungSik Choi, and Jong Sou Park. "Applying particle swarm optimization to prioritizing test cases for embedded real time software retesting." *Computer and Information Technology Workshops, 2008. CIT Workshops 2008. IEEE 8th International Conference on*. IEEE, 2008.

[2] Nagar, Reetika, et al. "Implementing test case selection and reduction techniques using meta-heuristics." *Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference-*. IEEE, 2014.

[3] Cavalcanti, Ana, Augusto Sampaio, and Jim Woodcook. *Testing techniques in software engineering*. Springer-Verlag Berlin Heidelberg, 2010.

[4] Wong, W. Eric, et al. "A study of effective regression testing in practice."*Software Reliability Engineering, 1997. Proceedings., The Eighth International Symposium on*. IEEE, 1997.

[5] Li, Zheng, Mark Harman, and Robert M. Hierons. "Search algorithms for regression test case prioritization." *IEEE Transactions on software engineering* 33.4 (2007): 225-237.

[6] Tallam, Sriraman, and Neelam Gupta. "A concept analysis inspired greedy algorithm for test suite minimization." *ACM SIGSOFT Software Engineering Notes* 31.1 (2006): 35-42.

[7] Mala, D. Jeya et al. "ABC tester—artificial bee colony based software test suite optimization approach." *International Journal of Software Engineering* 2009; 15-43.

[8] Singh, Yogesh, et al, "Test case prioritization using ant colony optimization." *ACM SIGSOFT Software Engineering Notes* 2009; 1-7.

[9] Yoo, Shin, and Mark Harman. "Pareto efficient multi-objective test case selection." *Proceedings of the 2007 international symposium on Software testing and analysis*. ACM, 2007.

[10] Simon, Dan, "Biogeography Based Optimization", *IEEE Transactions on Evolutionary Computation* 2008; 702-713.

[11] Mirjalili, Seyedali, Seyed Mohammad Mirjalili, and Andrew Lewis. "Grey wolf optimizer." *Advances in Engineering Software* 69 (2014): 46-61.

[12] Nagar, Reetika, et al. "Test case selection and prioritization using cuckoos search algorithm." *Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), 2015 International Conference on*. IEEE, 2015.

[13] Emary, E., et al. "Feature subset selection approach by gray-wolf optimization." *Afro-European Conference for Industrial Advancement*. Springer International Publishing, 2015.

[14] Sharma, Yatin, and Lalit Chandra Saikia. "Automatic generation control of a multi-area ST–Thermal power system using Grey Wolf Optimizer algorithm based classical controllers." *International Journal of Electrical Power & Energy Systems* 73 (2015): 853-862.

[15] Mustaffa, Zuriani, Yuhanis Yusof, and Siti Sakira Kamaruddin. "Application of Grey Wolf Optimizer for Time Series Forecasting."

[16] Korayem, L., M. Khorsid, and S. S. Kassem. "Using Grey Wolf Algorithm to Solve the Capacitated Vehicle Routing Problem." *IOP Conference Series: Materials Science and Engineering*. Vol. 83. No. 1. IOP Publishing, 2015.

[17] Shankar, K., and P. Eswaran. "Sharing a Secret Image with Encapsulated Shares in Visual Cryptography." *Procedia Computer Science* 70 (2015): 462-468.

[18] Mirjalili, Seyedali, et al. "Multi-objective grey wolf optimizer: A novel algorithm for multi-criterion optimization." *Expert Systems with Applications*47 (2016): 106-119.

[19] Yoo, Shin et al, " Using Hybrid algorithm for Pareto Efficient multi-Objective test suite Minimization", The journal of Systems and Software 2010; 689-701.

[20] Rothermel, Gregg, et al. "Prioritizing test cases for regression testing." *IEEE Transactions on software engineering* 27.10 (2001): 929-948.

[21] Elbaum, Sebastian, Alexey G. Malishevsky, and Gregg Rothermel. "Test case prioritization: A family of empirical studies." *IEEE transactions on software engineering* 28.2 (2002): 159-182.

[22] Gonzalez-Sanchez, Alberto, et al. "Prioritizing tests for software fault localization." *2010 10th International Conference on Quality Software*. IEEE, 2010.

[23] Elbaum, Sebastian, Alexey G. Malishevsky, and Gregg Rothermel.*Prioritizing test cases for regression testing*. Vol. 25. No. 5. ACM, 2000.

[24] Elbaum, Sebastian, Alexey Malishevsky, and Gregg Rothermel. "Incorporating varying test costs and fault severities into test case prioritization." *Proceedings of the 23rd International Conference on Software Engineering*. IEEE Computer Society, 2001.

[25] Mohapatra, Sudhir Kumar, and Srinivas Prasad. "Evolutionary Search Algorithms for Test Case Prioritization." *Machine Intelligence and Research Advancement (ICMIRA), 2013 International Conference on*. IEEE, 2013.

[26] Mirarab, Siavash, and Ladan Tahvildari. "A prioritization approach for software test cases based on bayesian networks." *International Conference on Fundamental Approaches to Software Engineering*. Springer Berlin Heidelberg, 2007.

[27] Kaur, Arvinder et al, " A Bee Colony Optimization Algorithm for Fault Coverage Based Regression Test Suite Priortization", *International Journal of Advanced Science and Technology* 2011; 3037-3046.

[28] Kaur, Arvinder, et al, "Hybrid Particle Swarm Optimization for Regression Testing", *International Journal on Computer Science and Engineering* 2012; 1815-1824.

[29] Ali Haidar, Aftab et al, " On the Fly Test Suite Optimization with Fuzzy Optimizer", *2013 11th International Conference on Frontiers of Information Technology,* pp. 101-106, 2013.

[30] Carlson, Ryan, Hyunsook Do, and Anne Denton. "A clustering approach to improving test case prioritization: An industrial case study." *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*. IEEE, 2011.

[31] Vivekanandan, K, et al, "Improving Regression Testing through Modified Ant Colony Algorithm on a Dependency Injected Test Pattern", *International Journal of Computer Science Issues* 2012; 593-600.

[32] Tahvili, Sahar, Mehrdad Saadatmand, and Markus Bohlin. "Multi-criteria test case prioritization using fuzzy analytic hierarchy process." *Proceedings of the 10th International Conference on Software Engineering Advances (ICSEA 2015)*. 2015.

[33] Kumar, Harish, et al, "A hierarchical System Test Case Prioritization Technique Based on Requirements", *13th Annual International Software Testing Conference* 2013;1-9.

[34] Tahat, L. et al, "Regression test suite prioritization using system models", *Software Testing, Verification and Reliability* 2012; 481-506.

[35] Srivatsava, Praveen Ranjan, B. Mallikarjun, and Xin-She Yang. "Optimal test sequence generation using firefly algorithm." *Swarm and Evolutionary Computation* 8 (2013): 44-53.

[36] Rothermel, Gregg, and Mary Jean Harrold. "A safe, efficient regression test selection technique." *ACM Transactions on Software Engineering and Methodology (TOSEM)* 6.2 (1997): 173-210.

[37] Gove, R, et al, "Identifying Infeasible GUI Test Cases Using Support Vector Machines and Induced Grammers", *IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops,* pp. 202-211, 2011.

[38] Roongruangsuwan, Siripong, and Jirapun Daengdej. "Test case reduction methods by using CBR." *International Workshop on Design, Evaluation and Refinement of Intelligent Systems (DERIS2010)*. 2010.

[40] Manas Gaur, Thesis, "*Biogeography based optimization for complex system*", M.Tech , Delhi Technological University, 2015

[41] Yang, Xin-She, and Suash Deb. "Cuckoo search via Lévy flights." *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*. IEEE, 2009.

[42] Kim, Jung-Min, and Adam Porter. "A history-based test prioritization technique for regression testing in resource constrained environments."*Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*. IEEE, 2002.

[43] Meshoul, Souham, and Mohamed Batouche. "Ant colony system with extremal dynamics for point matching and pose estimation." *Pattern Recognition, 2002. Proceedings. 16th International Conference on*. Vol. 3. IEEE, 2002.

[44] Elish, Karim O., et al, "Predicting defect-prone software modules using support vector machines", *Journal of Systems and Software,* pp. 649-660, 2008.

[45] Solanki, Kamna, "An Empirical Literature Study of Various Test Case Prioritization Techniques", *Software Engineering and Technology* 2014; 169-173.

[46] MacArthur, Robert H., and Edward O. Wilson. *Theory of Island Biogeography.(MPB-1)*. Vol. 1. Princeton University Press, 2015.

[47] Rarick, Rick, et al. "Biogeography-based optimization and the solution of the power flow problem." *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*. IEEE, 2009.

[48] Yang, Gelan, et al. "Automated classification of brain images using wavelet-energy and biogeography-based optimization." *Multimedia Tools and Applications* (2015): 1-17.

[49] Dinesh, Shinde, et al. "APPLICATION OF SIMULATED ANNEALING ALGORITHM (SA) FOR DESIGN OPTIMIZATION OF AUTOMOBILE SUSPENSION SYSTEM."

[50] Payne R. B., Sorenson M. D., and Klitz K., The Cuckoos, Oxford University Press, (2005)

[51] Ouaarab, Aziz, Belaïd Ahiod, and Xin-She Yang. "Discrete cuckoo search algorithm for the travelling salesman problem." *Neural Computing and Applications* 24.7-8 (2014): 1659-1669.

[52] Srivastava, Praveen Ranjan, et al. "An efficient optimization algorithm for structural software testing." *International Journal of Artificial Intelligence*™8.S12 (2012): 68-77.

[53] Yang, G.; Yang, J. (2015). "*Automated classification of brain images using wavelet-energy and biogeography-based optimization*". Multimedia Tools and Applications. doi:10.1007/s11042-015-2649-7

[54] Emary, E., et al. "*Feature Subset Selection Approach by Gray-Wolf Optimization.*" Afro-European Conference for Industrial Advancement. Springer International, 2015.

[55] Singh, Yogesh, Software Testing, Cambridge University Press, 2011.

[56] Yang, Xin-She. *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.